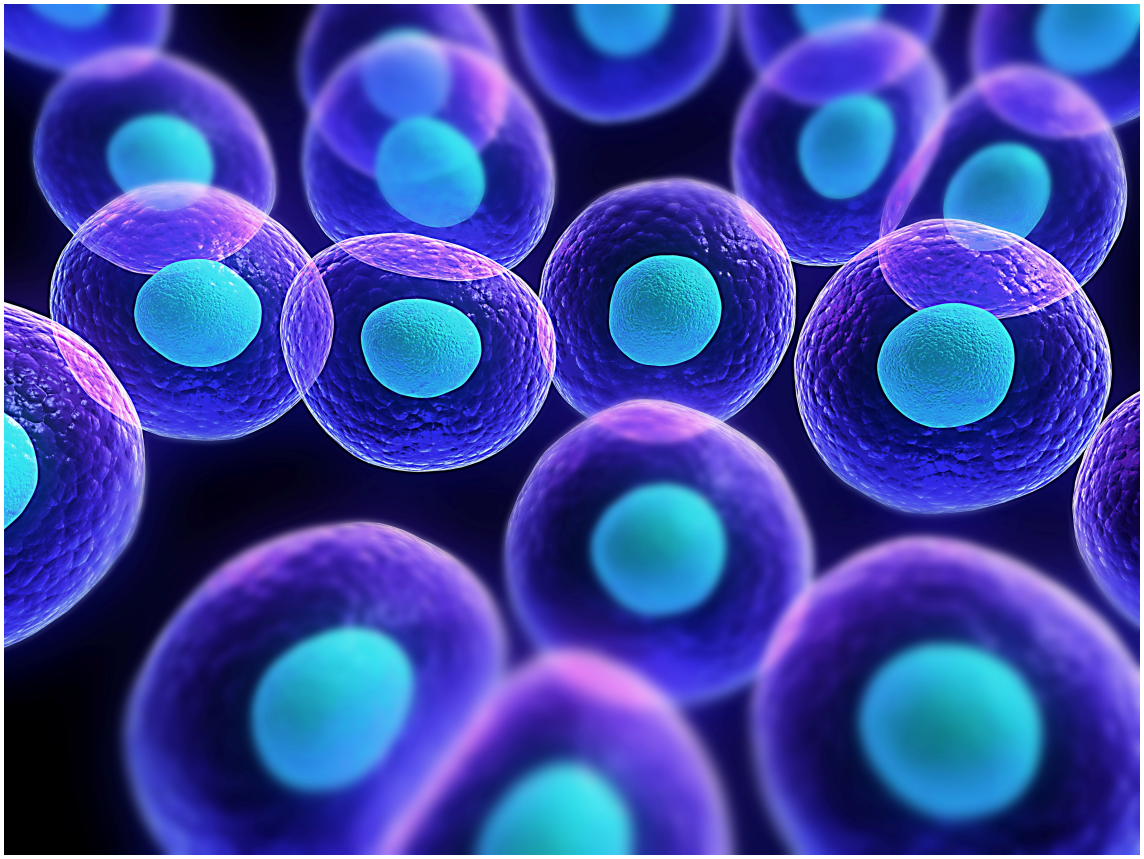
On Large-Scale Multiparty Computation with sub-linear Communication using Ephemeral Servers

Ph.D. Thesis



Credits: AI generated image by DALL-E 2 on cell apoptosis*

Anders Konring

Sepember 1, 2023

IT UNIVERSITY OF COPENHAGEN

Main Supervisor	**Bernardo David**	**IT University of Copenhagen**
Host Supervisor	Yuval Ishai	Technion - Israel Institute of Technology

*Human cells have an extremely sophisticated signalling system (pathways) forming a large-scale network. A biological mechanism named apoptosis (programmed cell death) causes the cell to split its state into units (DNA fragmentation) before self-destructing. This phenomenon is reminiscent of the actions of a machine participating in a proactive protocol by re-sharing its state before blowing up. The cell performs apoptosis to inhibit compromised (mutant) cells to divide and multiply. Over the course of a year the mass of self-destructed cells will approximately equal the full body weight of the human.

To my parents - Eva and Emil - teaching me kindness and curiosity.

Abstract

Secure Multiparty Computation (MPC) is a technology that enables a set of mutually-distrustful parties to securely compute a function on their inputs, without leaking any information about these inputs, beyond what is inferred from the output of the function. MPC is a useful tool in settings where it is unacceptable to rely on a trusted third-party to compute the function on behalf of the parties. This includes settings where data privacy is desirable (*e.g.* private auctions) but also in places where law and regulation (*e.g.* GDPR, CCPA, LGPD) would otherwise prevent such data from being subject to computation. The study of MPC protocols dates back to the seminal work of Andrew Yao (FOCS, 1986) and has garnered significant attention from cryptography researchers exploring new techniques, added efficiency, and inherent limitations of MPC protocols.

The emergence of large-scale permissionless networks such as Bitcoin and Ethereum has driven new interest in specific branches of MPC research aimed at combining the input-privacy of MPC with the resilience and scalability of modern blockchain networks. This would allow a set of limited clients to outsource a computation to "the blockchain" without revealing their inputs *aka.* MPC-as-a-Service. However, existing MPC protocols are designed to thrive when executed in the context of static and homogenous networks with high availability and low-latency communication and this makes them incompatible with the heterogenous and dynamic nature of permissionless networks. Moreover, the communication complexity of these protocols scales quadratically with the number of parties making them prohibitively expensive to execute at this scale.
Recent work of Gentry et al. (CRYPTO, 2021) proposed a model for MPC with the goal of identifying MPC protocols that overcome the above challenges. Such protocols are executed by a set of small randomly-selected committees and only allow each committee member to send a *single* message. Hence, the name: YOSO (*You Only Speak Once*) model. They also presented actual YOSO MPC protocols with statistical and computational security but left the question of establishing the underlying communication channels to future committee members largely unanswered.

This thesis provides a rigorous treatment of the problem of sending secret messages to future committee members. We provide a definitional framework around our main primitive - *Encryption to the Future* - and propose concrete constructions improving on existing protocols for establishing communication channels in the YOSO model. In addition, these existing protocols are not amenable to new efficient techniques for publicly verifiable resharing which is a key primitive when designing MPC protocols in the YOSO model. We observe that our framework naturally generalizes to settings where these efficient techniques are applicable by taking advantage of the underlying additive homomorphism of the encryption scheme. Thus, instead of relying on expensive generic non-interactive zero knowledge proofs for proving correct resharing, we utilize the algebraic structure and obtain extremely efficient proofs. Finally, we take a step back from communication in the YOSO model and ask a basic question of feasibility of perfect and fully secure protocols in the setting of standard MPC but with a specific layered interaction pattern. We answer this question in the affirmative and prove interesting implications for the YOSO model and other areas of dynamic MPC.

Abstrakt

MPC (*Multiparty Computation*) er en teknologi som giver forskellige parter muligheden for at beregne en given funktion på deres fælles data mens disse data samtidig holdes privat. Teknologien er især nyttig i scenarier hvor parterne ikke har tillid til at en betroet tredje-part modtager data og udfører beregningen på deres vegne. Det være sig både hvis privat beregning er ønskværdigt (f.eks. ved en auktion) men også når det er den eneste mulighed for at få lov at beregne på data (f.eks. GDPR-beskyttet data). Studiet af protokoller for MPC startede med det skelsættende arbejde af Andrew Yao (FOCS, 1986) og har siden da nydt megen opmærksomhed i kryptografi-kredse som herefter har forbedret protokollerne i et utal af forskellige retninger.

I de seneste år har større tilladelsesfrie netærk såsom Bitcoin og Ethereum skabt interesse for en specifik avart af MPC-forskning. Denne form for MPC bestræber sig på både at holde data privat men samtidig opnå samme skaleringsmuligheder og sikkerhedsegenskaber som det bagvedliggende netværk. Med andre ord, de benytter maskinerne i blockchain netværket til at eksekvere den private beregning. Eksisterende protokoller er dog ikke designet til opnå høj ydeevne i dette miljø. De er istedet skabt til at være effektive i en kontekst hvor netværket er stabilt, maskinerne har høj tilgængelighed og latensen er lav i kommunikationen mellem maskinerne. Ydermere, disse protokoller har kommunikations-kompleksitet der skalerer kvadratisk med størrelsen af netværket og er derfor praktisk talt umulige at eksekvere på disse store tilladelsesfrie netværk. Gentry et al. (Crypto, 2021) foreslog en MPC model der havde til hensigt at identificere protokoller som kan imødekomme ovenstående udfordringer. Disse protokoller er baseret på en teknik hvor en række tilfældigt-valgte komitéer udfører beregningen på vegne at det større netværk. Modellen blev præsenteret under navnet YOSO (*You Only Speak Once*) da alle maskinerne i denne model kun forventes at kunne sende én eneste besked på netværket. Forfatterne præsenterede også forskellige protokoller og påviste deres sikkerhed i denne nye YOSO model men spørgsmålet om hvordan disse maskiner kommunikerer fra én komité til den næste forblev i et stort omfang ubesvaret.

Denne afhandling bidrager med en grundig behandling af teknikker til at kommunikere med kommende komitéer. Til samme formål definerer vi et nyt krypterings-primitiv ved navn *Encryption to the Future* og bygger protokoller som opfylder kravende for denne definition og samtidig forbedrer ekisterende konstruktioner med samme formål. Vi bemærker at det ikke kun er kommunikationen som kan forbedres men også den måde den bliver brugt på. En vigtig sub-protokol i YOSO MPC - *publicly verifiable resharing* - har stor betydning for effektiviteten i YOSO MPC men eksisterende protokoller udnytter ikke de enorme fremskridt fra traditionelle (ikke-YOSO) MPC protokoller. Vi observerer at vores krypterings-primitiv faktisk også omfatter disse sub-protokoller og viser at dette kan lede til markant højere effektivitet i YOSO MPC protokoller. Til sidst bevæger vi os et smule væk fra kommunikations-delen og fokuserer på YOSO MPC beregningen givet at kommunikationen allerede er etableret. Ved hjælp af en vigtig indsigt; at YOSO MPC næsten er traditionel MPC blot med begrænset kommunikationsgraf formår vi at konstruere en ny YOSO MPC protokol med den højeste sikkerhedsgaranti og dermed besvare et essentielt spørgsmål omkring hvad der er muligt i YOSO modellen.

Acknowledgements

I would like to extend my gratitude to the people who provided me with the necessary background to make actual, albeit small, contributions in the area of theoretical cryptography and through intense discussions helped me refine my thinking.

My co-authors Matteo Campanelli, Hamidreza Khoshakhlagh, Jesper Buus Nielsen, Igancio Cascudo and Lydia Garms made the initial period of the PhD bearable while the pandemic was limiting most in-person interaction. I am grateful for having been given the opportunity to work with such wonderful researchers in such a hectic period.

During my time at Technion, I often found myself in intense discussions with Varun Narayanan. Through these discussions, he help me gain new insight in the in the realm of information theoretical cryptography and increased my clarity of thinking around proof construction and conducting research in general.

Working with Yuval Ishai and Eyal Kushilevitz at Technion taught me to ask good research questions. Often the hard part is not to come up with a good answer but, rather, to ask the right question - in its most condensed form. I experienced this first-hand with our work on Layered MPC. Yuval welcomed me into the research group at Technion and allowed me to present our work at CRYPTO'23 in Santa Barbara.

My supervisor Bernardo David has been invaluable through the whole PhD period. During the first year with COVID-19, Bernardo was incredibly helpful in forming ties with other like-minded researchers. He is not only a great teacher but also a good person and he has a passion for cryptography research which is contagious. I have enjoyed every minute of our discussions in his office.

Finally, my partner Amalie has kept me sane throughout the whole journey and has been my bedrock through ups and downs.

Thank you all.

Contents

Part I

Introduction

Chapter 1

Background

In January 2008 the first practical application of Secure Multiparty Computation (MPC) took place by facilitating a live sugar beet auction in Denmark [BCD+08]. This, somewhat, famous event in the cryptography community was a culmination of research conducted since Yao's garbled circuits [Yao86] in 1986. One year later, in January 2009, another ingenious idea was proposed when Satoshi Nakamoto published a paper [Nak08] describing the simple longest chain-idea that underpins the novel consensus mechanism which applications like Bitcoin and Ethereum [W+14] are build on. Fast-forward a decade and the two technologies have created a synergistic effect where the rise of blockchain serves as an enabler for many unresolved issues in MPC [GG17, CGJ+17] and, similarly, decades of research of MPC protocols has been applied to meet some of the challenges in the realm of blockchains [Lin17, GG18, RA23]. In this thesis we focus on a specific area of cryptography which considers "blockchain-friendly" MPC protocols for general computation and the communication within those protocols. This section provides the necessary background and motivates the contributions presented in later sections.

1.1 Multiparty Computation

At a high level, an MPC protocol allows a set of parties to compute a function on their joint inputs in a secure way. Security means, roughly speaking, that even when some parties misbehave, they can neither disrupt the output to honest parties (correctness), nor can they obtain more information from the computation than what is contained in their individual inputs and outputs (privacy). Misbehaving parties are represented by adversary types; *semi-honest* (passive) and *malicious* (active) where the semi-honest adversary merely observes the state and messages received at corrupted parties but must follow the protocol specification and the malicious adversary is, moreover, free to arbitrarily change the messages on behalf of the corrupted parties.

Seminal results from the 1980s [Yao86, GMW87] showed that, under standard cryptographic assumptions, *any* multi-party functionality can be securely computed in the presence of a polynomial bounded and *semi-honest* adversary corrupting arbitrarily many parties[1]. In [GMW87], it was proven that arbitrarily many corruptions by a *malicious* adversary can only be tolerated if we are willing to relax the correctness property,

[1] This is also known as security with dishonest majority. Obviously, if all parties are corrupted the security of the protocol is meaningless.

give up *fairness*[2] and settle for *security-with-abort* instead of *guaranteed output delivery* (G.O.D.).

The situation is different in the realm of information-theoretic (IT) security where there are no restrictions on the adversary's computational power. Here, [BGW88,CCD88] showed that IT security is possible only in the setting of honest super-majority where the malicious adversary may corrupt at most $t < n/3$ ($t < n/2$, if the adversary is semi-honest). The protocol in [BGW88] enjoys *perfect full security* meaning there are no probability of error (*perfect security*) and the output is guaranteed to arrive at the honest parties (G.O.D.).

In [RB89], Rabin and Ben-Or relaxed the requirement of perfect security and instead allowed for a negligible probability of error (*statistical security*) and, moreover, assumed the availability of a broadcast channel[3]. In this setting, they proved that any function can be computed with honest majority *i.e.* if a malicious adversary may corrupt at most $t < n/2$. The same result was also achieved by [Bea92a] but with better efficiency.

Often information theoretic protocols are preferred since they tolerate a computationally more capable adversary and in general demonstrate good concrete efficiency. Moreover, protocols in the IT-setting are known to enjoy strong composability guarantees [KLR06] as opposed to their computationally secure counterparts where composability without a trusted setup (such as a common reference string (CRS)) is generally impossible [CF01].

In relation to the protocol's degree of composability, a crucial distinction is made between two types of protocol execution. The *stand-alone* setting is where the protocol is executed only once in isolation. All the results above were initially obtained by analyzing the protocols in the stand-alone setting. And the *concurrent* setting that assumes the protocol is run in a network concurrently with other, arbitrary protocols. There a countless examples of protocols where security does not remain intact when executed in the concurrent setting [Lin03].

Protocols in the stand-alone setting may be analyzed using *game-based* approach, where the security of a primitive is modelled as one or more games between a challenger and the adversary, or using the *simulation* paradigm where a protocol emulates a trusted third party (ideal functionality). [Can00] proved that the set of secure protocols in the stand-alone setting is closed under a natural composition operation, namely, the so-called *subroutine substitution* operation. This security framework is also known as *sequential composability* (*e.g.* [AL17]). To capture security of protocols in the concurrent setting, the Universal Composability (UC) framework [Can01] is the method of choice. Proving a protocol UC-secure is generally considered best-practice since the power of the adversary resembles reality in a network with concurrent protocols but it also requires a keen eye for designing the ideal functionality. UC secure protocols exist essentially for any functionality in case of honest majority or with a trusted setup (fx CRS or public key infrastructure (PKI)). Thus, achieving the same broad feasibility results as in the stand-alone model. However, if no honest majority and no trusted setup is available there are large classes of functionalities that simply cannot be UC-realized [CKL06].

[2] Fairness refers to the MPC property that either *every* party receives its output or no one does. Fairness is impossible (in general) for protocols with dishonest majority in the malicious setting [Cle86].

[3] The assumption of broadcast is necessary due to the impossibility result of [FLM85].

Many protocols (including the protocols behind the feasibility results above) assume that each party in the MPC is able to provide input and obtain output from the computation. The *client-server* model [CDI05, DI06] refines the standard MPC model and separates the parties into *clients*, that can provide input and obtain output, and *servers*, that are responsible for the computation[4]. This allows for protocol analysis that depends only on the number of computing servers effectively decoupling the clients from affecting the communication complexity and security level of the protocol. One direct approach of achieving security in the client/server model is to have clients share their inputs to all servers $(P_1, \ldots, P_n)$, allowing the servers to compute the result using an appropriate protocol ([GMW87, BGW88, CCD88, RB89]) and send the result back to the clients. This approach, however, incurs a large overhead when the number of clients is small compared to the number of servers. In fact, the communication complexity of such a protocol is at least polynomial in n and is clearly infeasible to execute in large-scale networks where the number of parties can reach thousands (or maybe millions).

One way to improve efficiency in such large-scale networks is by forming random sublinear-size server committees to do computation on behalf of the network. This idea dates back to [Bra85]. Surprisingly, [GIOZ17] showed that even in the statistical setting it is actually possible[5] to tolerate semi-honest adaptive (see Section 1.3) corruption of up to $\approx (1 - \sqrt{0.5})n$. In a nutshell, the protocol's random coins and inputs induce a graph of point-to-point communication channels between committee parties which in turn can be traversed by the adversary to extend its corruption set. In [GIOZ17], this is modeled as a *probabilistic* adversary that assigns to each subset of parties a probability that it gets corrupted. Using this model, they prove that the bound is tight *i.e.* it is impossible to tolerate $t > (1 - \sqrt{0.5})n$ in this setting. Their protocol relies on the use of indirection (so-called intermediaries) that ensures that the random parties are able to erase their state before the adversary can meaningfully corrupt them. Interestingly, the same asymptotic bound also crops up in the seminal work of [BGG+20] in the context of MPC on permissionless blockchains. As we shall see next, this is no coincidence since many consensus protocols perform the same kind of random sampling albeit weighted by the stake in the system.

1.2 Permissionless Consensus

The literature on consensus is vast and dates back to the 1980s. The seminal work of [PSL80] was aimed at designing algorithms to achieve interactive consistency[6] (consensus) in settings where faulty parties may display arbitrary behavior. Later, a colorful metaphor by Lamport [LSP82] gave name to these *Byzantine Fault Tolerant* (BFT) algorithms. Designing BFT algorithms has subsequently become a central topic in distributed computing. The choice of communication model is central to the limitations in the design-space of BFT algorithms. [PSL80] showed that achieving consensus in the synchronous model is possible if and only if at most $t < n/3$ of the n parties are byzantine

[4] Note that a party can be both a client and server. A setting where all parties are both client and servers, again, yields the standard MPC model

[5] Assuming erasures—the party's ability to erase its past state (see Section 1.3).

[6] The first BFT algorithms were suggested for coordination of interacting processes in airplanes. Robert Shostak suggested the phrase interactive consistency before BFT became the used label.

but the algorithm must run in at least $t+1$ rounds [FL81, FLM86]. In the asynchronous setting, a (deterministic) BFT algorithm that solves consensus (even for $t = 1$ crash-failure) must run infinitely [FLP85]. This gave rise to the *partial synchronous* model of [DLS84] that is a now popular sweet-spot between synchrony and asynchrony (*e.g.* [LAM98, CL02]). Using signature-chains (PKI-assumption), [DS83] showed that consensus is possible with honest majority ($t < n/2$) in the synchronous model but still subject to the lower-bound of $t + 1$ rounds.

Until 2008, the consensus problem was only studied in the *permissioned* setting where all involved parties are known to each other at the outset of the protocol execution. What differentiate the most prominent blockchain protocol Bitcoin [Nak08] from the above is that it operates in the *permissionless* setting, *i.e.* it is a protocol for establishing consensus over an unknown set of parties that anyone can join. It is clear that classical results for the permissioned setting will not carry over to the permissionless setting directly. As an example consider the lower bound of [DS83] that establishes a minium of $t + 1$ rounds for solving consensus but if the number of faulty parties t is unbounded then consensus is seemingly impossible. The unbounded set of parties also gives the adversary the chance to gain disproportional large influence in the system by creating large number of pseudonymous identities (sybil attack). Mechanisms for obtaining *sybil resistance*, therefore, lies at the heart of any permissionless consensus algorithm. The two most well-known categories of *sybil resistance* are proof-of-work (PoW) and proof-of-stake (PoS). A key ingredient in a sybil resistance mechanism for a permissionless network is that a party's influence on the protocol execution is determined by its access to some limited resource. In PoW, all parties (miners) engage in a lottery where each party is selected proportional to its access to compute while in PoS each party is selected to act with probability proportional to its stake in the system. Importantly, this is very different from regular cryptographic protocols where the all parties have the same "power".

Roughly, Bitcoin [Nak08] can be seen as an example of a PoW sybil resistance mechanism combined with the *longest-chain* rule. This combination makes consensus possible in the permissionless setting if more than half[7] of the computing power is controlled by honest nodes. Many frameworks [PSs17, GKL15] are dedicated to the analysis of this combination and it remains the most prominent in the category of PoW. The other sybil resistance mechanism in the permissionless setting is PoS which is, arguably, a much more heterogenous category. Notable examples of PoS based on the longest-chain rule is [DGKR18, DPS19] while other approaches have combined PoS with classic BFT-like protocols ([GHM+17, YMR+19]).

PoS sybil resistance mechanisms come in different shapes and forms where all allow a party (or a set of parties) to identify as the leader for a given time slot. Arguably, the simplest being weighted round robin or "follow the satoshi" that deterministically chooses a single leader as a function of the publicly-accessible randomness. The downside of this approach is that it does not provide any privacy for the leader and, thus, it can be subject to denial of service (DoS) attacks before it has carried out its protocol duties. *Single Secret Leader Election* (SSLE) is an interesting new area of research [BEHG20, CFG22] that addresses this specific concern. Still, the approach based on a verifiable random func-

[7] Note, neither PoW nor PoS consensus circumvent the [FLP85]-impossibility result since both rely on strong setup assumptions.

tion (VRF) ([CM19, GHM+17, DGKR18]) remains the most well-understood sybil resistance mechanism even if the protocol has to take into account the variance[8] in the number of selected leaders.

1.3 The Adaptive Setting

Earlier, in the outline of MPC and permissionless consensus protocols, we assumed that the adversary chooses the set of corrupted parties at the outset of the protocol (*static* adversary). This is clearly an unrealistic assumption. On the other hand, the last example above illustrated that even single leader election becomes much harder when dealing with an *adaptive* adversary that can mount DoS attacks (or even fully corrupt) parties during the protocol execution. However, notice that in the case of a private sybil resistance mechanism even an adaptive adversary cannot do better than randomly guessing.[9]

Algorand [CM19, GHM+17] popularized the private VRF-based leader election mechanism (cryptographic sortition) that together with a specific BFT algorithm constitute a PoS consensus protocol secure against an adaptive adversary. In fact, this consensus protocol can withstand an even more powerful adversary that can corrupt *every* party in the protocol eventually but is subject to a corruption rate of $t < n/3$ in each round.[10] This is known as the *mobile adversary* [OY91] and it is strictly stronger than its adaptive counterpart that may also adaptively corrupt a party at any given round but is stuck with this decision until the end of the protocol execution. As such, the mobile adversary can dynamically move between parties and this greatly limits the design-space of protocols that can withstand it. The specific BFT algorithm of Algorand is called BA* and is a modification of the classic protocol of [FM88]. The resulting protocol has the property of being *player replaceable* which means that a new set of parties (committee) can execute each round of the protocol. Indeed, this property combined with the private leader election are necessary if the protocol is to remain resilient towards a mobile adversary as a party that plays a key role in multiple rounds are an easy target for the mobile adversary.

Ostrovsky and Yung [OY91] were not only considering the problem of designing resilient (proactive) BFT algorithms but rather the harder problem of long-running MPC where the protocol may carry secret state that must not fall in the hands of the adversary. In this setting, an execution is divided in rounds that are grouped into epochs. The adversary can "move" at the onset of every epoch by choosing a new set of parties to corrupt and remains static for the remainder of the epoch. Former corrupted parties are "rebooted" into a clean initial state (or, equivalently, update their internal state and securely erase past state). In [OY91], it is proven that there exists a fully secure proactive MPC protocol in the presence of an active mobile adversary with 1-round epochs but allowing only a small constant fraction of corrupted parties. Subsequent works [HJKY95, ADN06, BELO15,

[8] The VRF-based approach can be parameterized for a single unique leader *in expectation* but each slot may yield zero or multiple leaders. The underlying consensus protocol needs to cater to that variance.

[9] This is under the assumption that an adversary cannot withdraw a message upon corrupting the leader and that the adversary generally is unable to "front-run" the leader. The latter issue can be partially addressed by the use of VDFs [BBBF18].

[10] This is a simplification. Actually, [CM19] considers the amount of stake owned by honest parties should be more than $2/3$.

ELL20] explored more efficient protocols with other security guarantees under further restrictions on the mobile adversary, but still fell short of 1-round epochs or achieving the optimal corruption threshold ($t < n/3$) of [BGW88].

1.4 YOSO MPC

Departing from player replaceability and anonymous committees in permissionless settings, the notion of You Only Speak Once (YOSO) MPC (introduced in [GHK+21]) takes proactive security one step further, by having a freshly elected anonymous committee of parties execute each round of the protocol. As an extra restriction, parties are only allowed to send messages once (*i.e.* when they execute their role in the protocol). With the introduction of the YOSO model, the authors explicitly distinguished between two role-related aspects to solve. The first is the *role assignment* which deals with the sending of messages to parties selected to perform future roles of a protocol while hiding the identities of such parties. The other aspect is *role execution* which focuses on the execution of the specific protocol that runs on top of the role assignment mechanism, *i.e.*, what messages are sent to which roles and what specification the protocol implements. In [GHK+21], the YOSO model was introduced for studying role execution between abstract roles which can each speak only once. Later, these roles are mapped to physical machines using role assignment. The work of [GHK+21] showed that given role assignment in a synchronous model, any well-formed ideal functionality can be implemented in the YOSO model with statistical security against malicious adaptive corruption of $t < n/2$ machines.

In the YOSO model, the fact that each round is executed by anonymous parties elected at random makes the adversary inherently probabilistic. In particular, the threshold of $t < n/2$ for secure YOSO MPC of [GHK+21] does not apply for constant n. Apart from not allowing for worst-case corruption due to variance in sampling, the YOSO model relies on a subtle but strong assumption, namely, that the ideal channels can reach a role in an arbitrary future round. In [GHK+21], the authors also presented an efficient, computationally secure protocol based on [CDN01] and on-demand Beaver triple [Bea92a] generation relying on a trusted setup. However, recent work [KRY22, BDO22] improved on this state of affairs by achieving G.O.D. in a constant number of rounds without relying on this trusted setup.

Fluid MPC [CGG+21] is a variant of this model where role assignment is abstracted away and where parties cannot access ideal channels to parties in an arbitrary later round. Also, the model may be configured such that parties may act in more than one round before being substituted. However, the results presented in [CGG+21] fall short of full security (they do not achieve G.O.D.). An interesting variation was shown in SCALES [AHKP22], which allows for special clients who provide an input and receive an output to act in more than one round (while server committees may only act once), focusing on protocols with computational security.

1.5 Role Assignment

Cryptographic protocols traditionally rely on secure channels among parties whose identities are publicly known. However, while knowing parties' identities makes it easy to construct secure channels, it also makes it easy for an adaptive (or mobile) adversary to corrupt parties as a protocol execution proceeds.

Role assignment was first introduced in [BGG+20] and further developed in [GHK+21]. This task consists of sending a message to an abstract role R at a given point in the future. A role is just a bit-string describing an abstract role, such as R ="party number j in round sl of the protocol Γ". Behind the scenes, there is a mechanism that samples the identity of a random party P_i and associates this machine to the role R. Such a mechanism allows anyone to send a message m to R and have m arrive at P_i chosen at some point in the future to act as R. A crucial point is: no one should know the identity of P_i even though P_i learns that it is chosen to act as R.

The approaches proposed in [BGG+20, GHK+21, GHM+21] for realizing role assignment all use an underlying Proof-of-Stake (PoS) blockchain (*e.g.* [GHM+17, DGKR18]). On a blockchain, a concrete way to implement role assignment is to sample a fresh key pair $(\mathsf{sk}_\mathsf{R}, \mathsf{pk}_\mathsf{R})$ for a public key encryption scheme, post $(\mathsf{R}, \mathsf{pk}_\mathsf{R})$ on the blockchain and somehow send sk_R to a random P_i without leaking the identity of this party to anyone. Once $(\mathsf{R}, \mathsf{pk}_\mathsf{R})$ is known, every party has a target-anonymous channel to P_i and is able to encrypt under pk_R and post the ciphertext on the blockchain. Notice that using time-lock puzzles (or similar notions) is not sufficient for achieving this notion, since only the party or parties elected for a role should receive a secret message encrypted for that role, while time-lock puzzles allow any party to recover the message if they invest enough computing time.

A shortcoming of the approaches of [BGG+20, GHK+21, GHM+21] is that, besides an underlying blockchain, they require an auxiliary committee to aid in generating $(\mathsf{sk}_\mathsf{R}, \mathsf{pk}_\mathsf{R})$ and selecting P_i. In the case of [BGG+20], the auxiliary committee performs cheap operations but can adversarially influence the probability distribution with which P_i is chosen. In the case of [GHK+21, GHM+21], the auxiliary committee cannot bias this probability distribution but must perform very expensive operations (using Mix-Nets or FHE; see also Section 2.1). Moreover, these approaches have other caveats:

- they are incompatible with efficient techniques for publicly proving that encrypted secret shares are valid.

- they can only be used to select P_i to act as R according to a probability distribution *already known* at the time the auxiliary committee outputs $(\mathsf{R}, \mathsf{pk}_\mathsf{R})$. Hence, they only allow sending messages to future committees that have been *recently* elected.

1.6 Extreme Resilience leading to Improved Efficiency

A natural side-effect of a protocol being resilient towards a powerful mobile adversary, is that the same protocol can tolerate a high churn rate; any party may join the network, execute a single round as a role in the protocol and then leave again. As such, YOSO MPC epitomizes the permissionless character of large-scale networks such as blockchain environments. But these networks may have thousands or even millions of parties making even the most efficient adaptive MPC protocols prohibitively expensive to execute in practice. Fortunately, the role assignment mechanism described above helps avoid the quadratic dependency on network size N that arise in the communication complexity of standard protocols. This is done by sampling small, anonymous committees of size $n \ll N$ to carry out the computation on behalf of a network. Provided that the size of the sampled committee is large enough, this guarantees that even if the adversary may corrupt $\Omega(N)$ parties, the honest/dishonest-distribution of committee members will stay within the tolerated threshold with all but negligible probability. This makes the YOSO model attractive, not only as a framework for analyzing extremely resilient MPC protocols, but also as a means to achieve MPC protocols with communication that scales sub-linearly with the number of parties N and, thus, are efficient both asymptotically and in practice.

Chapter 2

Contributed Work

In this thesis we explore the emerging domain of large-scale MPC using ephemeral servers. As discussed above, this field of research offers several new and interesting open problems but at the same time the domain is rooted in a long history of research in MPC and consensus protocols. In this section we first provide a brief overview of the contributions and then progress to a more detailed treatment of each publication and the corresponding related work.

Encryption to the Future In Encryption to the Future (Chapter 6) we establish a new paradigm for sending secret messages to future (possibly anonymous) committees with applications to role assignment. Existing protocols for role assignment [BGG+20, GHM+21, GHK+21] suffer from sub-optimal corruption threshold, use of expensive operations or generally rely on interaction with auxiliary committees. Moreover, other protocols that use role assignment as an ideal resource often assume non black-box access to the underlying encryption keys for authentication which strays from the attractive notion of having access to a simple point-to-point channel to a future committee member. Finally, the same ideal resource is parameterized with a *future horizon* which, roughly, decides how long channels are "alive". But the ephemeral nature of nodes in this setting induces existing role assignment protocols to support only small future horizons even if allowing for interaction with auxiliary committees.

Taking a step back from existing approaches, we define the primitive Encryption to the Future (EtF) with the aim of obtaining *non-interactive* solutions to the role assignment problem. We define security in the context of a PoS blockchain execution with an underlying lottery predicate selecting the receiving party. We distinguish between "near-future" (encryption to the current winner—ECW) and "far-future" EtF depending on whether the lottery is conducted wrt. known or unknown stake distribution, respectively. We improve existing role assignment protocols by constructing ECW protocols *without* interaction using standard assumptions and show a general EtF protocol that support receivers in the far future but with minimal use of auxiliary committees. We show that role assignment can be realized using ECW and we define a central primitive for this; a corresponding authentication mechanism we label *Authentication from the Past* (AfP).

YOLO YOSO In this work we focus on designing publicly verifiable secret sharing schemes (PVSS) in the YOSO model [GHK+21]. At the heart of any PVSS is a mechanism

that allows everybody to verify that shares have been correctly distributed—a share validity check. Standard efficient PVSS protocols require that the identity of the receivers of shares are known in order to check share validity which is clearly incompatible with the YOSO model. Moreover, while existing PVSS protocols in the YOSO model do share structured messages that must be verified, the underlying role assignment protocols do not take advantage of this structure and protocols instead resolve to expensive generic non-interative zero knowledge proof of encrypted shares validity.

In YOLO YOSO we construct two PVSS schemes: the first (HEPVSS) is a generic PVSS scheme that achieves efficient proofs of encrypted shares validity from any IND-CPA additively homomorphic encryption scheme. The other DDH-based construction (DHPVSS) has a highly optimized proof based on the so-called SCRAPE test [CD17] and incurs a communication overhead of only 2 group elements. Both constructions also exhibit efficient proofs of correct decryption used when reconstructing the secret and can be extended to additionally prove that the shared value is the one corresponding to a public ciphertext (proof of resharing).

Apart from the efficiency gains in regular PVSS protocols, we also bring these improvements into the realm of the YOSO model. We do this by first introducing a new an simple ECW scheme that improves on the existing schemes by having ciphertexts that are linear in the number of receivers (not in the network size). The ECW scheme requires a setup phase where a mixnet shuffles ephemeral encryption keys, however, the scheme has the immediate downside of linking a encryption key to a party identity during authentication. Thus, the setup has to either support multiple authentications (re-usable) or rely on a (pre-processed) set of shuffled encryption keys. Finally, we instantiate the PVSS constructions on our newly developed ECW scheme along with a resharing protocol allowing for parties to efficiently "keeping a secret alive"—a core component of YOSO MPC protocols [BGG+20].

Layered MPC The seminal work of [OY91] introduced the concept of a mobile adversary that is able to compromise all parties but is limited to to a threshold of parties at any given time. The authors also put forth a protocol with full (perfect and G.O.D.) security in the presence of such an adversary albeit tolerating only a small constant fraction of corrupted parties at any given time instead of the optimal corruption threshold of BGW [BGW88] ($t < n/3$). In this work we investigate the possibility of building MPC protocols achieving full security against an adaptive rushing adversary while maintaining the optimal corruption threshold of $t < n/3$.

Recent work on MPC with ephemeral committees investigate the concept of having a freshly elected committee of parties execute each round of the protocol with the restriction that they can only speak once (when they execute their part in the protocol). Protocols in the YOSO model [GHK+21] are generally considered secure against a mobile adversary but the model relies on access to ideal target-anonymous channels and, moreover, the random selection of parties results in a probabilistic adversary which is incompatible with settings where n and t are constant. The model of Fluid MPC [CGG+21] revolves around the same theme of outsourcing the computation to a set of ephemeral committees and while the model does not dictate communication through ideal target-

anonymous channels, the protocols are in the statistical setting and only enjoy security-with-abort (not G.O.D.).

As a tool in our investigation, we define layered MPC which captures the most stringent setting in the intersection of the mobile adversary and the YOSO models. We provide a formalization of this model and show that layered MPC protocols can be analyzed within well established frameworks such as the real/ideal world paradigm [Can00, Gol09] and Universal Composability [Can01]. We show that a secure layered MPC protocol is also secure against a maximally mobile adversary [OY91], that moves after every round. And with the tool of layered MPC we answer in the affirmative the main question of achieving full security while maintaining optimal corruption threshold $t < n/3$ (see Section 2.3).

2.1 Encryption to the Future—Chapter 6

In [CDK+22] (ASIACRYPT 2022) we investigate solutions to the role assignment problem (see Section 1.5). Taking a step back from specific solutions to this problem, we strive to obtain *non-interactive* solutions to encrypting to a future role with IND-CPA security without the aid of an auxiliary committee. To this end, we introduce a new paradigm for sending secret messages to future (possibly anonymous) unknown committees. At the heart of this paradigm is a cryptographic primitive named Encryption to the Future (EtF). We define this primitive in the context of a blockchain, which determines the passing of time, and a lottery, that selects parties to receive a secret message at some point in the future.

Using this paradigm we progress to address the main shortcomings of existing approaches to role assignment:

- Interaction with Auxiliary Committees: They require an auxiliary committee to aid in generating ephemeral keys and selecting a machine to perform a role. In the case of [BGG+20], the auxiliary committee performs cheap operations but can adversarially influence the probability distribution with which a party is chosen for a given role. In the case of [GHM+21], the auxiliary committee cannot bias this probability distribution but must perform very expensive operations (Mix-Nets or FHE).

- Communication restricted to "Near-Future": They can only be used to select a party to act as a role according to a probability distribution *already known* at the time the auxiliary committee outputs the ephemeral key pair. Hence, both [BGG+20, GHM+21] only allow sending messages to future committees that have been *recently* elected. We explicitly consider this weaker setting—where we want to communicate with a "near-future" committee (i.e., whose distribution is known)—and dub it "Encryption to the Current Winner[1]" (ECW).

We improve on solutions relying on interaction with an auxiliary committee and shed light on the hardness of achieving a fully ("far-future") non-interactive solution. We also discuss how to extend our approach to IND-CCA2 security and how to allow winners of a role to authenticate themselves when sending a message, achieving both goals using standard assumptions.

Apart from defining this primitive and showing constructions based on previous works, we propose constructions based on new insights and investigate limits of EtF in different scenarios. Our general construction for EtF works by lifting a weaker primitive, namely encryption for the aforementioned "near-future" setting, or ECW. Before providing further details, we summarize our contributions as follows:

Definition of EtF: A definition for the notion of Encryption to the Future (EtF) in terms of an underlying blockchain and an associated lottery scheme that selects parties in the future to receive messages for a role. We study the strength of EtF as a primitive and prove that a non-interactive EtF scheme allowing for encryption towards parties

[1]The word "winner" here refers to the party who is selected to perform a role according to the underlying lottery of the PoS blockchain (see remainder of introduction).

selected at arbitrary points in the future implies a flavor of witness encryption for NP over a blockchain (referred to as BWE).

Non-Interactive ECW: A novel construction of Encryption to the Current Winner (ECW), *i.e.* EtF where the receiver of a message is determined by the *current* state of the blockchain, which can be instantiated *without* interaction with auxiliary committees from standard assumptions via a construction based on generic primitives.

EtF with minimal use of auxiliary committees: A transformation from ECW to EtF through an auxiliary committee holding a *small* state, i.e., with communication complexity *independent* of plaintext size $|m|$ (in contrast to [BGG+20, GHK+21, GHM+21] where a committee's state grows with $|m|$).

Role Assignment from ECW: An application of ECW as a central primitive for realizing role assignment in protocols that require it (*e.g.* [BGG+20, GHK+21, GHM+21]).

Our EtF notion arguably provides a useful abstraction for the problem of transferring secret states to secret committees. Our ECW construction is the first primitive to realize role assignment without the need for an auxiliary committee. Moreover, building on new insights from our EtF notion and constructions, we show the first protocol for obtaining role assignment with no constraints on when parties are chosen to act as the role. While our protocol uses auxiliary committees, it improves on previous work by only requiring a communication complexity *independent* of the plaintext length. We elaborate on our results, discussing the intuition behind the notion of EtF, its constructions and its fundamental limits.

Related Work

Type	Scheme	Communication	Committee?	Interaction?
ECW	CaBKaS [BGG+20]	$O(1)$	yes	yes
	RPIR [GHM+21]	$O(1)$	yes	yes
	cWE (MS-NISC) (Sec. 6.3.2)	$O(N)$	no	no*
	cWE (GC+OT) (Sec. 6.3.2)	$O(N)$	no	no*
EtF	IBE (Sec. 6.6)	$O(1)$	yes	yes
	WEB [GKM+22]	$O(M)$	yes	yes
	Full-fledged WE	$O(1)$	no	no

Table 2.1: The column "Committee?" indicates whether a committee is required. The column "Communication" refers to the communication complexity in terms of the number of all parties N, and the number of plaintexts (called deposited secrets in [GKM+22]) M of a given fixed length. We denote by an asterisk non-interactive solutions that require sending a first reusable message during the initial step.

Encryption to the Current Winner (ECW). We recall that ECW is an easier setting than EtF: both the stake distribution and the randomness extracted from the blockchain are static and known at the time of encryption. This means that all of the parameters except

the secret key of the lottery winner are available to the encryption algorithm. We now survey works that solved this problem and compare them to our solutions:

- "Can a Blockchain Keep a Secret?" (CaBKaS) [BGG+20]. The work of [BGG+20] addresses the setting where a dynamically changing committee (over a public blockchain) maintains a secret. The main challenge in order for the committee to *securely* reshare its secret can be summarized as: how to select a small committee from a large population of parties so that everyone can send secure messages to the committee members without knowing who they are? The solution of [BGG+20] is to select the "secret-holding" committee by having another committee, a "nominating committee", that nominates members of the former (while the members of the nominating committee are self-nominated). One can see the nominating committee as a tool providing the ECW functionality. A major caveat in such a solution, however, is that to guarantee an honest majority in the committees, [BGG+20] can only tolerate up to $t \approx 0.29n$ corrupted parties. This is because corrupted nominators can always select corrupted parties, whereas honest nominators may select corrupted parties by chance. We can improve this through our non-interactive ECW: we can remove the nominating committee and just let the current committee ECW-encrypt their secret shares to the roles of the next committee.

- "Random-Index PIR" (RPIR) [GHM+21]. The constraint on corruption ratio of [BGG+20] was subsequently solved in [GHM+21] via random-index private information retrieval (RPIR). RPIR allows a client to retrieve a random index from a database in such a way that the servers holding the database do not learn what index was retrieved. The solution of [GHM+21] consists in running an RPIR protocol with a database holding the public keys of all parties and having parties in a committee execute the client using MPC, outputting re-randomized versions of the public keys output by RPIR. While RPIR improves on [BGG+20] (not requiring a nominating committee and tolerating up to 1/2 of corrupted parties), its constructions are inefficient, either based on Mix-Nets or Fully Homomorphic Encryption (FHE). The construction based on Mix-Nets uses k shufflers, where k is the security parameter, and has an impractical communication complexity of $O(nk^2)$, where n is the number of public keys that each shuffler broadcasts. The FHE-based construction gives a total communication complexity of $O(k^3)$ where $O(k)$ is the length of an FHE decryption share.

WE over commitments (cWE). Benhamouda and Lin [BL20] defined a type of witness encryption, called "Witness Encryption for NIZK of Commitments". In their setting, parties first commit to their private inputs once and for all. Later, an encryptor can produce a ciphertext so that any party with a committed input that satisfies the relation (specified at encryption time) can decrypt. More accurately, who can decrypt is any party *with a NIZK showing that the committed input satisfies the relation*. The authors construct this primitive based on standard assumptions in asymmetric bilinear groups.

In our work, we generalize the encryption notion in [BL20], formalize it as cWE and finally use it to construct ECW. While the original construction of [BL20] fits the definition of cWE, we observe it is an overkill for our application. Specifically our setting does not require NIZKs to be involved in encryption/decryption. We instead give more efficient instanti-

ations based on two-party Multi-Sender Non-Interactive Secure Computation (MS-NISC) protocols and Oblivious Transfer plus Garbled Circuits.

Encryption to the Future (EtF). The general notion of EtF is significantly harder to realize than ECW (as we show in Section 6.7). Below we discuss natural ideas to obtain EtF. They can be seen as illustrating two extremes where our approach (Section 6.6) lies in the middle.

- NON-INTERACTIVE: Using Witness Encryption [GGSW13]: One trivial approach to realize EtF is to use full-fledged general Witness Encryption [GGSW13] (WE) for the arithmetic relation $\mathcal{R}$ being the lottery predicate such that the party who holds a winning secret key sk can decrypt the ciphertext. However, constructing a general witness encryption scheme [GGSW13] which we can instantiate reliably is still an open problem. Existing constructions rely on very strong assumptions such as multilinear maps, indistinguishability obfuscation or other complexity theoretical conjectures [BIOW20]. The challenges in applying this straightforward solution are not surprising given our result showing that EtF implies a flavor of WE.

- INTERACTIVE: Multiple Committees and Continuous Executions of ECW: A simple way to achieve an interactive version of EtF is to first encrypt secret shares of a message towards members of a committee that then re-share their secrets towards members of a future anonymous committee via an invocation of ECW (in our instantiations or those in [BGG+20] and [GHM+21]). This is essentially the solution proposed in CaBKaS [BGG+20] where committees interact in order to carry a secret (on the blockchain) into the future. Notice that, for a fixed security parameter and corruption ratio, the communication complexity of the protocol executed by the committee in this solution depends on the plaintext message length. On the other hand, for a fixed security parameter and corruption ratio, the communication complexity of our committee-based transformation from ECW to EtF is *constant*.

Other works. Using blockchains in order to construct non-interactive primitives with game-based security has been previously considered in [GG17]. Other approaches for transferring secret state to future committees have been proposed in [GKM+22], although anonymity is not a concern in this setting. On the other hand, using anonymity to overcome adaptive corruption has been proposed in [GGJ+15], although this work considers anonymous channels among a fixed set of parties.

2.2 YOLO YOSO—Chapter 7

In [CDGK22] (ASIACRYPT 2022) we address the issue of constructing simple ECW schemes amenable to efficient publicly verifiable secret (re)sharing (PVSS) protocols. Our contributions are summarized as follows:

Simple Encryption to Future (ECW): We construct a simple ECW scheme based on a mixnet and an additively homomorphic public key encryption scheme. Our scheme requires a setup phase where a mixnet is used but this setup can be either done once and reused for multiple times (using our reusable AFP) or preprocessed so that future encryptions can be done non-interactively. Our ECW ciphertexts have size linear *only in the number of parties who open them*.

Reusable Private Authentication from the Past (AFP): We show how to reuse our ECW setup even when a party performs multiple rounds of AFP, *i.e.* proving that it was selected to decrypt a given ECW ciphertext. This scheme guarantees that the adversary cannot predict which parties can decrypt future ECW ciphertexts while keeping the setup constant size.

Generic Efficient PVSS: We construct a generic PVSS protocol with efficient proofs of encrypted shares validity from any IND-CPA additively homomorphic encryption scheme with an efficient proof of decryption correctness without any generic zero knowledge proofs, which we call HEPVSS. This general result sheds new light on the construction on efficient PVSS schemes.

New PVSS with Minimal Overhead: Moreover, we introduce a new PVSS construction named DHPVSS with *constant-size proof of sharing correctness* which, as far as we know, is the first PVSS to achieve this. More precisely, the PVSS communicates only the n encrypted shares (which are one group element each) and two field elements for the proof. This may be of independent interest for other applications, such as randomness beacons.

Efficient PVSS for Anonymous Committees based on ECW: We instantiate our PVSS constructions based on our ECW and AFP schemes along with a protocol for resharing a secret towards a future random anonymous committee. This allows for parties to keep a secret alive, which is a core component of YOSO MPC.

Related Work

Encryption to the Future: Although, the protocol of [CDK+22] is constructed from simple tools like garbled circuits and oblivious transfer (after a setup phase), each encryption still requires communication and computational complexities linear in the total number of parties. The construction of [CDK+22] relies on a relaxation of Witness Encryption called Witness Encryption over Commitments (cWE), where one can encrypt a message towards the holder of an opening of a commitment to a valid witness of an NP relation. More specifically, we are interested in the case of Encryption to the Current Winner (ECW),

where the data needed to determine the party selected to perform a role is already in the underlying blockchain (but still does not reveal who the party is). In order to realize ECW, each party commits to a witness of a predicate showing they win a lottery for the current parameter. A party encrypting towards a role simply encrypts the message towards the party who has such a committed witness to winning the lottery for a current parameter. A party who wins can decrypt the message encrypted towards the role using their witness. They can perform *Authentication from the Past* (AfP) on a message by doing a signature of knowledge on that message using their lottery winning witness.

The ETF constructions of [CDK+22] suffer from a major drawback: every encryption towards an anonymously selected party has communication complexity $O(n\kappa)$ where n is the *total* number of parties and κ is the security parameter. Even if preprocessing is allowed, these constructions still require the sender to publish n cWE ciphertexts or to have the eligible receivers perform a round of anonymous broadcast that is only usable for a single encryption. On the other hand, the AfP constructions only have $O(\kappa)$ communication complexity.

PVSS Compatibility: A drawback in current role assignment [BGG+20, GHM+21, CDK+22] is that they are not amenable to publicly verifiable secret (re)sharing. Both in YOSO proactive secret sharing [BGG+20] and YOSO MPC [GHK+21], the committees executing each round of the protocol do not simply send unstructured messages but shares of a secret that must be verified. While this can be done via generic non-interactive zero knowledge proofs of encrypted shares validity, such a solution incurs very high computational and communication costs.

Publicly Verifiable Secret Sharing (PVSS): An integral part of YOSO protocols is having each committee perform PVSS towards the next committee. A PVSS scheme allows for any party to check that an encrypted share vector is valid. A number of PVSS constructions are known [Sta96, FO98, Sch99, BT99, RV05, HV09] that different techniques for proving that a vector of encrypted shares are valid shares of a given secret. Recently, the SCRAPE [CD17] and ALBATROSS [CD20] PVSS schemes have significantly improved on the complexity of such schemes by making the share validity check and reconstructions procedures cheaper than previous works. While these works are based on number theoretical assumptions, a recent work has shown how to efficiently build PVSS from lattice based assumptions [GHL21]. These works are not fit for the YOSO model because they require the parties to know the identities (or rather the public keys) of the parties receiving the shares when checking share validity, precluding (re)sharing towards anonymous parties. A key part of this work is that we explore the fact that the share validity check of SCRAPE can be modified to work regardless of the public keys used to encrypt the shares.

2.3 Layered MPC—Chapter 8

In [DKI+23] (CRYPTO 2023) we investigate the feasibility of perfect full security against a rushing adaptive adversary with optimal threshold ($t < n/3$). This question was originally settled in BGW [BGW88] but, inspired by the mobile adversary of [OY91] and recent line of work on MPC with dynamic committees [GHK+21, CGG+21, AHKP22], we ask again the same question albeit in the more challenging setting of MPC with dynamic committees.

As a tool in this investigation, we define layered MPC which captures the most stringent setting in the intersection of the mobile adversary and the YOSO models. In layered MPC, parties communicate through a directed layered graph of d layers corresponding to each protocol round. Each round is executed by a unique set of n parties sitting at a layer, which is disjoint from all other sets of parties in other layers. Parties in one layer can only receive messages from parties in the immediately previous layer and send messages to the parties in the immediately next layer. We consider an active, adaptive, rushing adversary that corrupts up to t out of n parties in each layer. We write (n, t, d)-layered MPC as shorthand for a layered MPC protocol with d layers (*i.e.* rounds) of n parties out of which t may be corrupted. We provide a formalization of this model and show that layered MPC protocols can be analyzed within well established frameworks such as the real/ideal world paradigm [Can00, Gol09] and Universal Composability [Can01].

Layered MPC is similar to maximally-Fluid MPC [CGG+21] with parties only executing one round. We show that a secure layered MPC protocol is also secure against a maximally mobile adversary [OY91], that moves after every round. In comparison to YOSO [GHK+21], layered MPC imposes stronger restrictions on honest parties, who cannot receive a message from a party in an arbitrary past committee or send a message to a party in an arbitrary future committee. Moreover, similar to Fluid MPC, the adversary is not restricted to probabilistic corruptions but is limited to corrupting t out n parties in each layer, allowing for threshold-optimal protocols.

Main Results. In Section 8.3 we construct basic primitives that help realize layered VSS based on CNF[2] (replicated) secret sharing. We present a nontrivial adaptation of a VSS protocol of Gennaro et al. [GIKR01] to the layered setting. The main challenge is to eliminate the repeated interaction between the parties and the dealer, which is not possible in the layered setting. While CNF-based protocols scale exponentially with n, they are simpler than their Shamir-based counterparts that we will present next, and can have efficiency advantages for small values of n, especially when settling for computational security.

Theorem 2.1 (CNF-Based Layered VSS). *For any n, t such that $t < n/3$, and $d \geq 5$, there exists an (n, t, d)-layered MPC protocol realizing CNF-VSS. For $d = O(1)$ and secrets of length ℓ, the protocol requires $\ell \cdot 2^{O(n)}$ bits of communication, counting both point-to-point messages and broadcast. When settling for* computational *security with perfect correctness and using a black-box PRG with seed length λ, there is a protocol with $\lambda \cdot 2^{O(n)} + O(n\ell)$ bits of communication.*

[2] In CNF-based secret sharing, the secret is first split into $\binom{n}{t}$ additive shares–a share r_T for each set $T \subset [n]$ of size t–and party i receives all shares r_T such that $i \notin T$.

In Section 8.4 we build on the above VSS protocol to obtain a *general* layered MPC protocol based on CNF secret sharing. The protocol applies to layered arithmetic circuits, in which each layer of the circuit only takes inputs from the previous layer. Every circuit of depth D can be converted to a layered circuit with D layers, incurring at most a quadratic but typically (nearly) linear overhead to the circuit size. Building on a constant-round protocol from [DI05], in Section 8.7 we describe how to amortize the overhead of CNF secret sharing by settling for computational security.

Theorem 2.2 (CNF-Based Layered MPC)**.** *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite ring, with D layers and M gates. Then, for any $t < n/3$, there is an $(n, t, O(D))$-layered MPC protocol for f. The communication consists of $2^{O(n)} \cdot M$ ring elements. Alternatively, settling for* computational *security with perfect correctness and using a black-box PRG with seed length λ, there is a $(n, t, O(1))$-layered MPC protocol for a Boolean circuit (i.e., the ring is $\mathcal{F}_2$) with M gates with $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ bits of communication.*

While the CNF-based protocols are relatively simple and have concrete efficiency benefits for small values of n, they do not yield a general feasibility result that scales polynomially with n. In Section 8.5 we establish such a result using (the bivariate version of) Shamir's secret-sharing scheme.

Theorem 2.3 (Efficient Layered MPC)**.** *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D layers and M gates. Then, for any $t < n/3$, there is a polynomial-time $(n, t, O(D))$-layered MPC protocol for f. More concretely, the communication consists of $M \cdot O(n^9)$ field elements.*

Further, in Section 8.6, we present a computationally secure, efficient layered protocol that achieves G.O.D. against adversaries who can corrupt $t < n/2$ parties in each layer.

Theorem 2.4 (Efficient Layered MPC for $t < n/2$)**.** *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $t < n/2$, there is an $(n, t, O(D))$-layered MPC protocol for f assuming non-interactive linearly-homomorphic equivocal commitments. The communication complexity is $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^5)$ field elements + $M \cdot O(n^{10} \cdot \lambda)$ bits over the broadcast channels, where λ is the security parameter.*

Proactive MPC. The original concept of proactive MPC put forward by [OY91] considered an adversary that has the ability to corrupt a fresh set of parties in *every* round of the protocol. We refer to such an adversary as maximally mobile. This notion is formally defined in Definition 8.2, while protocols that can thwart such an adversary are called maximally proactive. We show that a secure layered MPC protocol is a maximally proactively secure protocol. We also remark on an alternate and stronger notion of maximal adversary in Remark 8.2, against which perfectly secure VSS and MPC are impossible with the optimal threshold of $t < n/3$ corruptions in each layer. This allows us to extend our security

analysis from the layered to the proactive setting. Definitions 8.4 and 8.5 defines maximally Proactive Secret Sharing and MPC and we obtain the following threshold-optimal result by combining Theorem 2.3 and Lemma 8.1.

Corollary 2.1 (Perfectly Secure Maximally Proactive MPC)**.** *Let f be an n-party functionality computed by a layered circuit C over a field $\mathbb{F}$, with D layers. Then, for $t < n/3$, there is an efficient maximally proactive MPC protocol computing f in $r = O(D)$ rounds.*

Secure Message Transmission and Broadcast. Sending a message to a party that acts in an arbitrary future round is a recurring problem in settings such as layered MPC. In YOSO [GHK+21] it is circumvented by assuming target-anonymous channels, an ideal resource that allows a party in round r to send a message to a party who is elected to perform a certain role in round $r' > r + 1$ without learning its identity. We take steps to obtain a similar primitive (although without anonymity guarantees) by relying only on the parties in the layered graph to carry the message forward, despite our much more restrictive interaction pattern that precludes such communication. In Section 8.3.1.2 we provide a thorough analysis of an important primitive in layered MPC called *Future Messaging*. The functionality f_{FM} is described in Section 8.3.1.2 and presented in Fig. 8.2. Future Messaging takes as input a message m from a sender in $\mathcal{L}_0$ and, if the sender is honest, the message m arrives at the recipient. In the context of layered MPC this primitive is close to an instance of 1-way Secure Message Transmission (SMT) over a directed graph. We show that it is possible to self-compose this primitive to carry a message from a sender in $\mathcal{L}_0$ to a designated receiver in $\mathcal{L}_d$ for $d > 1$. The following theorem characterizes our construction.

Theorem 2.5 (Restatement of Theorem 8.1)**.** *For any $d > 0$, any n and t where $t < n/3$, and message domain M, there exists a protocol Π_{FM} that realizes f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ with perfect t-security and communication complexity $O(n^{\lceil \log d \rceil} \log |M|)$.*

Using the layered protocol for Shamir VSS and resharing, which we construct building on Future Messaging, we can make the dependence of the communication cost of Future Messaging on d linear. This is achieved by having the sender verifiably secret the message using VSS and then reshare it repeatedly until reaching the layer previous to that of the receiver, at which point the shareholders of the value can reveal the message to the receiver by transferring all its shares. Communication cost of VSS and of resharing across a constant number of layers is $\mathsf{poly}(\lambda)(n)$, making the communication of Future Messaging linear in d.

The layered model allows for layer-to-layer broadcast: any party in $\mathcal{L}_a$ may broadcast to parties in $\mathcal{L}_{a+1}$. It turns out that this assumption is necessary, since we prove that deterministic broadcast in the setting of layered MPC is possible only if $t = 0$. Our analysis is presented in Section 8.8, where we cast the result of [Gar94] to the setting of layered MPC and obtain the following result.

Theorem 2.6. *Deterministic perfect Broadcast in the setting of layered MPC is possible iff $t = 0$.*

This limitation can be overcome by the use of randomization. Several works achieve broadcast in the honest-majority setting with overwhelming probability after a number of rounds that is linear in the security parameter, without setup tolerating $t < n/3$ corruptions [FM85], and with different types of setup tolerating $t < n/2$ corruptions [KK06, FG03, FLZL21, GGLZ22].

These protocols can be ported to the layered setting at the cost of decreasing the corruption threshold by a factor that is linear in the security parameter. This is done by naively porting the protocol to the layered setting after ensuring that the parties 'persist' across all the layers by simply forwarding the view of each party to their counterpart in the next layer. When the adversary corrupts t' parties in each layer, by the end of the protocol, the adversary would corrupt at most $t = t' \cdot O(\kappa)$ parties executing different party roles, for security parameter κ. If the total number of corruptions that the original protocol tolerates is bounded by $t < n/3$ (resp. $t < n/2$), we have that the ported protocol remains secure. Obtaining the optimal corruption threshold $t' < n/3$ without setup, or $t' < n/2$ with setup, for broadcast is beyond the scope of this paper.

Related Work

Results for Maximally Proactive MPC with Dynamic Committees					
Functionality	Reference	Level	Security	Complexity	Threshold
Future Messaging	Section 8.3.1.2	perfect	full	$\mathsf{poly}(\lambda)(n)$	$t < n/3$
VSS	[BGG+20]	computational	full	$\mathsf{poly}(\lambda)(n)$	$t < n/4^*$
	Section 8.4.2	perfect	full	$2^{O(n)}$	$t < n/3$
	Section 8.5	perfect	full	$\mathsf{poly}(\lambda)(n)$	$t < n/3$
MPC	[GHK+21] (YOSO)	statistical	full (w/setup†)	$\mathsf{poly}(\lambda)(n)$	$t < n/2^*$
	[CGG+21] (Fluid)	statistical	w/abort	$\mathsf{poly}(\lambda)(n)$	$t < n/2$
	[OY91]	perfect	full	$\mathsf{poly}(\lambda)(n)$	$t < n/d$
	Section 8.4.4	perfect	full	$2^{O(n)}$	$t < n/3$
	Section 8.5	perfect	full	$\mathsf{poly}(\lambda)(n)$	$t < n/3$
	Section 8.6	computational	full	$\mathsf{poly}(\lambda)(n)$	$t < n/2$

Table 2.2: Protocols realizing primitives in the most extreme proactive settings. (*protocol security relies on the adversary only doing probabilistic corruption, †assumes access to ideal target-anonymous channels for future messaging)

Proactive Secret Sharing (PSS). PSS protocols aim at solving the problem that shares learned by the adversary are compromised forever by resharing the secret periodically. The *static group* setting where resharing is done among the same set of parties is considered in [HJKY95, CH01, ADN06, BELO15]. However, this is often insufficient since it assumes a world where a server never fails to the extend that it cannot recover again. The setting of *dynamic groups* where resharing is done towards a different (possibly disjoint) set of parties is considered in [DJ97, WWW02, ELL20]. Finally, proactive techniques in asynchronous settings have been treated in [CKLS02, SLL10].

Permissionless Networks. In the context of permissionless networks where parties are allowed to join and leave as they wish, the *dynamic group* property has taken on a new meaning. The notion of player replaceability (where the set of parties get replaced in every round) has previously been studied in the context of consensus primitives [Mic17, CM19, PS17, BKLZL20]. The recent focus on this setting spurred new interest in (dynamic) proactive techniques [MZW+19, GKM+22]. Particularly interesting, is the definition of evolving committee secret sharing [BGG+20] that places the responsibility of keeping a tolerable corruption threshold on the protocol designer.

Maximally PSS and MPC with Dynamic Committees. Recently, a number of works [GHM+17, GHK+21, CGG+21, AHKP22] have considered extreme settings with dynamic committees, where each round of a protocol is executed by a new set of parties considering maximally mobile (or even adaptive) adversaries. In YOSO [GHK+21], an ideal mechanism guarantees that a set of anonymous parties is selected at random to execute each round, effectively limiting the adversary to probabilistic corruptions. Hence, YOSO is incompatible with settings where n and t are constant. Moreover, parties have access to ideal target-anonymous channels allowing for communication to *any* party in the future. Hence, results in the YOSO model do not directly translate to our setting even if we settle for non-optimal corruption thresholds, as YOSO protocols may crucially rely on the ability to send messages across many layers. For example, in the information theoretical signature protocol of [GHK+21, Section 3.3], a cut-and-choose mechanism is realized assuming that a sender can commit to a set of message authentication codes (MACs) by sending them directly to a receiver, after which verifiers broadcast random subsets of keys, which the receiver uses to check these MACs. The security of this technique crucially relies on the fact that using ideal target-anonymous channels guarantees that the sender cannot changes the MACs sent to the user *after* the verifiers announce the checking keys. This technique does not work in the layered MPC setting with our weaker Future Messaging protocol, which does not commit a corrupted sender to the messages it transmits to future layers.

Closest to layered MPC is Fluid MPC [CGG+21] in its most extreme configuration (fully fluid), where parties can execute a single round of the protocol and immediately leave but are not necessarily selected anonymously and at random. Curiously, one of the goals of Fluid MPC is maintaining a small state complexity. In particular, the computation and communication of each committee in Fluid MPC is independent of the size of the circuit. While this is attractive, we do not make any such claims and we also only consider already layered circuits[3]. Finally, a crucial difference is that the known protocols for Fluid MPC only enjoy security-with-abort while we aim for full security.

While the use of an arbitrary interaction pattern in layered MPC is similar to [HIJ+16], our focus is on a specific interaction pattern capturing extreme cases of MPC with dynamic committees and a maximally mobile adversary.

[3] The inherent issue with state complexity originates from a common misconception (see fx [DEP21]) that *any* general arithmetic circuit can be transformed into a layered circuit with same depth and only linear overhead in width.

Chapter 3

Organization

The remainder of this thesis is divided into three parts: Part II presents all the definitions that underpin the work presented in the chapters ahead. Here, Chapter 4 discusses the definitions related to permissionless blockchains that is used throughout the thesis and Chapter 5 lists all the cryptographic primitives that are used in the publications. This part of the thesis offers little discussion and can easily be skipped (and used as a reference point) when going through the remainder of the thesis.

In Part III, each chapter represents a published paper. As such, the results presented in these chapters completely overlap with the full versions available online. Only few minor editorial changes have been made to enhance readability and to present the publications in a coherent way.

Finally, in Part IV, we first discuss (Chapter 9) the thesis from a birds-eye perspective and shed light on the motivation to pursue the problems in this area of research. We then conclude by exploring a few directions for future work in Section 9.2.

Part II

Preliminaries

Chapter 4

Blockchain Fundamentals

4.1 Proof-of-Stake (PoS) Blockchains

In this work we rely on PoS-based blockchain protocols. In such a protocol, each participant is associated with some stake in the system. A process called leader election encapsulates a lottery mechanism that ensures (of all eligible parties) each party succeeds in generating the next block with probability proportional to its stake in the system. In order to formally argue about executions of such protocols, we depart from the framework presented in [GG17] which, in turn, builds on the analysis done in [GKL15] and [PSs17]. We invite the reader to re-visit the abstraction used in [GG17]. We present a summary of the framework in Section 4.2 and then continue with the main properties we will use in the remainder of this work. Moreover, we note that in [GG17] it is proven that there exist PoS blockchain protocols with the properties described below, *e.g.* Ouroboros Praos [DGKR18].

4.2 Blockchain Protocol Execution.

Let the blockchain protocol $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ be guarded by a validity predicate V. The algorithms can be described as follows:

- $\mathsf{UpdateState}(1^\lambda) \rightarrow \mathsf{bst}$ where bst is the local state of the blockchain along with metadata.
- $\mathsf{GetRecords}(1^\lambda, \mathsf{bst}) \rightarrow \mathbf{B}$ outputs the longest sequence $\mathbf{B}$ of valid blocks (wrt. V).
- $\mathsf{Broadcast}(1^\lambda, m)$ Broadcast the message m over the network to all parties executing the blockchain protocol.

An execution of a blockchain protocol Γ^V proceeds by participants running the algorithm $\mathsf{UpdateState}^V$ to get the latest blockchain state, $\mathsf{GetRecords}$ to extract the ledger data structure from a state and $\mathsf{Broadcast}$ to distribute messages which are added to the blockchain if accepted by V. An execution is orchestrated by an environment $\mathcal{Z}$ which classifies parties as either honest or corrupt. All honest parties executes $\Gamma^V(1^\lambda)$ with

empty local state bst and all corrupted parties are controlled by the adversary $\mathcal{A}$ who also controls network including delivery of messages between all parties.

- In each round all honest parties receive a message m from $\mathcal{Z}$ and potentially receive incoming network messages delivered by $\mathcal{A}$. The honest parties may do computation, broadcast messages and/or update their local states.
- $\mathcal{A}$ is responsible for delivering all messages sent by honest parties to all other parties. $\mathcal{A}$ cannot modify messages from honest parties but may delay and reorder messages on the network.
- At any point $\mathcal{Z}$ can communicate with adversary $\mathcal{A}$ or use GetRecords to retrieve a view of the local state of any party participating in the protocol.

The result is a random variable $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ denoting the joint view of all parties (i.e. all inputs, random coins and messages received) in the above execution. Note that the joint view of all parties fully determines the execution. We define the view of the adversary as $\mathsf{view}_{\mathcal{A}}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$ and the view of the party P_i as $\mathsf{view}_{P_i}(\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda))$. If it is clear from the context which execution the argument is referring to, then we just write view_i. We assume that it is possible to take a snapshot i.e. a view of the protocol after the first r rounds have been executed. We denote that by $\mathsf{view}^r \leftarrow \mathsf{EXEC}_r^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$. Furthermore, we can resume the execution departing from this view and continue until round $\tilde{r}$ resulting in the full view including round $\tilde{r}$ denoted by $\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma^V}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$.

We let the function $\mathsf{stake}_i = \mathsf{stake}(\mathbf{B}, i)$ take as input a local blockchain $\mathbf{B}$ and a party P_i and output a number representing the stake of party P_i wrt. to blockchain $\mathbf{B}$. Let the sum of stake controlled by the adversary be $\mathsf{stake}_{\mathcal{A}}(\mathbf{B})$, the total stake held by all parties $\mathsf{stake}_{\mathsf{total}}(\mathbf{B})$ and the adversaries relative stake is $\mathsf{stake\text{-}ratio}_{\mathcal{A}}(\mathbf{B})$. We also consider the PoS-fraction $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell)$ as the amount of unique stake whose proof is provided in the last ℓ mined blocks. More precisely, let $\mathcal{M}$ be the index i corresponding to miners P_i of the last ℓ blocks in $\mathbf{B}$ then

$$\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell) = \frac{\sum_{i \in \mathcal{M}} \mathsf{stake}(\mathbf{B}, i)}{\mathsf{stake}_{\mathsf{total}}}$$

A note on corruption. For simplicity in the above execution we restrict the environment to only allow static corruption while the execution described in [PSs17] supports adaptive corruption with erasures.

A note on admissible environments. [PSs17] specifies a set of restrictions on $\mathcal{A}$ and $\mathcal{Z}$ such that only compliant executions are considered and argues that certain security properties holds with overwhelming probability for these executions. An example of such a restriction is that $\mathcal{A}$ should deliver network messages to honest parties within Δ rounds.

4.3 Blockchain Properties.

We recall that running a protocol Γ^V with appropriate restrictions on $\mathcal{A}$ and $\mathcal{Z}$ will yield certain compliant executions $\mathsf{EXEC}^{\Gamma^V}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ where some security properties will hold with overwhelming probability. An array of prior works, including [GKL15,PSs17], have converged towards a few security properties that characterizes blockchain protocols. These include *Common Prefix* or *Chain Consistency*, *Chain Quality* and *Chain Growth*. From these basic properties, a number of stronger properties were derived in [GG17]. Among them, is the *Distinguishable Forking* property. We denote by $\mathbf{B}^{\lceil \ell}$ the chain (sequence of blocks) $\mathbf{B}$ where the last ℓ blocks have been removed and if $\ell \geq |\mathbf{B}|$ then $\mathbf{B}^{\lceil \ell} = \epsilon$ and we write $\mathbf{B}_1 \preceq \mathbf{B}_2$ if $\mathbf{B}_1$ is a prefix of $\mathbf{B}_2$.

Definition 4.1 (Common Prefix). *Let $\kappa \in \mathbb{N}$ be the common prefix parameter. The chains $\mathbf{B}_1$, $\mathbf{B}_2$ possessed by two honest parties P_1 and P_2 in slots $\mathsf{sl}_1 < \mathsf{sl}_2$ satisfy $\mathbf{B}_1^{\lceil \kappa} \preceq \mathbf{B}_2$.*

Definition 4.2 (Chain Growth). *Let $\tau \in (0, 1]$, $s \in \mathbb{N}$ and let $\mathbf{B}_1$, $\mathbf{B}_2$ be as above with the additional restriction that $\mathsf{sl}_1 + s \leq \mathsf{sl}_2$. Then $\mathsf{len}(\mathbf{B}_2) - \mathsf{len}(\mathbf{B}_1) \geq \tau s$ where τ is the speed coefficient.*

Definition 4.3 (Chain Quality). *Let $\mu \in (0, 1]$ and $\kappa \in \mathbb{N}$. Consider any set of consecutive blocks of length at least κ from an honest party's chain $\mathbf{B}_1$. The ratio of adversarial blocks in the set is $1 - \mu$ where μ is the quality coefficient.*

Stake Contribution Property. At a high level, the sufficient stake contribution property states that after sufficiently many rounds, the total amount of proof-of-stake in mining the ℓ most recent blocks is at least β fraction of the total stake in the system.

Definition 4.4 (Sufficient Stake Contribution). *Let* suf-stake-contr *be the predicate such that* $\mathsf{suf\text{-}stake\text{-}contr}^{\ell}(\mathsf{view}, \beta) = 1$ *iff for any round $r \geq \ell$, and any party i in* view *that is honest at round r with blockchain $\mathbf{B}$, we have* $\mathsf{u\text{-}stakefrac}(\mathbf{B}, \ell) > \beta$. *A blockchain protocol Γ has $(\beta(\cdot), \ell_0(\cdot))$-sufficient stake contribution property with adversary $\mathcal{A}$ in environment $\mathcal{Z}$, if there is a negligible function* $\mathsf{negl}(\cdot)$ *such that for any $\lambda \in \mathbb{N}$, $\ell \geq \ell_0$, it holds that*

$$\Pr\left[\mathsf{suf\text{-}stake\text{-}contr}^{\ell}(\mathsf{view}, \beta(\lambda)) = 1 \,\middle|\, \mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda)$$

Bounded Forking Property. Roughly speaking, the bounded forking property requires that no efficient adversary can create a sufficiently long fork so that its total amount of proof of stake is higher than a certain threshold. In more detail, it states that for property parameters α, ℓ_1, ℓ_2, the proof-of-stake fraction in the last ℓ_2 blocks in any adversarially created fork of length at least $\ell_1 + \ell_2$ should not be more than α.

Definition 4.5 (Bounded Stake Forking). *Let* bd-stake-fork *be the predicate such that* $\mathsf{bd\text{-}stake\text{-}fork}^{(\ell_1,\ell_2)}(\mathsf{view}, \alpha) = 1$ *iff for any round $r \geq \tilde{r}$, and any pair of parties i, j in*

view *such that* i *is honest at round* r *with blockchain* $\mathbf{B}$ *and* j *is corrupt in round* $\tilde{r}$ *with blockchain* $\tilde{\mathbf{B}}$*, if there exists* $\ell' \geq \ell_1 + \ell_2$ *such that* $\tilde{\mathbf{B}}^{\lceil \ell'} \preceq \mathbf{B}$ *and for all* $\tilde{\ell} < \ell'$, $\tilde{\mathbf{B}}^{\lceil \tilde{\ell}} \not\preceq \mathbf{B}$*, then* u-stakefrac$(\tilde{\mathbf{B}}, \ell' - \ell_1) \leq \alpha$.
A blockchain protocol Γ *has* $(\alpha(\cdot), \ell_1(\cdot), \ell_2(\cdot))$*-bounded stake forking property with adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$*, if there exists negligible functions* negl$(\cdot)$ *and* $\delta(\cdot)$ *such that for any* $\lambda \in \mathbb{N}$, $\ell \geq \ell_1(\lambda)$, $\tilde{\ell} \geq \ell_2(\lambda)$*, it holds that*

$$\Pr\left[\text{bd-stake-fork}^{(\ell,\tilde{\ell})}(\text{view}, \alpha(\lambda) + \delta(\lambda)) = 1 \;\middle|\; \text{view} \leftarrow \text{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})\right] \geq 1 - \text{negl}(\lambda)$$

Definition 4.6 (Distinguishable Forking)**.** *A blockchain protocol* Γ *satisfies* $(\alpha(\cdot), \beta(\cdot), \ell_1(\cdot), \ell_2(\cdot))$*-distinguishable forking property with adversary* $\mathcal{A}$ *in environment* $\mathcal{Z}$*, if there exists negligible functions* negl$(\cdot)$*,* $\delta(\cdot)$ *such that for every* $\lambda \in \mathbb{N}, \ell \geq \ell_1(\lambda), \tilde{\ell} \geq \ell_2(\lambda)$ *it holds that*

$$\Pr\left[\begin{array}{c} \alpha(\lambda) + \delta(\lambda) < \beta(\lambda) \wedge \\ \text{suf-stake-contr}^{\tilde{\ell}}(\text{view}, \beta(\lambda)) = 1 \wedge \\ \text{bd-stake-fork}^{(\ell,\tilde{\ell})}(\text{view}, \alpha(\lambda) + \delta(\lambda)) = 1 \end{array} \;\middle|\; \text{view} \leftarrow \text{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})\right] \geq 1 - \text{negl}(\lambda)$$

4.4 Blockchain Lotteries

4.4.1 Blockchain Structure.

A genesis block $B_0 = \{(\text{Sig.pk}_1, \text{aux}_1, \text{stake}_1), \ldots, (\text{Sig.pk}_n, \text{aux}_n, \text{stake}_n), \text{aux}\}$ associates each party P_i to a signature scheme public key Sig.pk_i, an amount of stake stake_i and auxiliary information aux_i (*i.e.* any other relevant information required by the blockchain protocol, such as verifiable random function public keys). A blockchain $\mathbf{B}$ relative to a genesis block B_0 is a sequence of blocks $B_1, \ldots, B_n$ associated with a strictly increasing sequence of slots $\text{sl}_1, \ldots, \text{sl}_m$ such that $B_i = (\text{sl}_j, H(B_{i-1}), \text{d}, \text{aux}))$. Here, sl_j indicates the time slot that B_i occupies, $H(B_{i-1})$ is a collision resistant hash of the previous block, d is data and aux is auxiliary information required by the blockchain protocol (*e.g.* a proof that the block is valid for slot sl_j). Each party participating in the protocol has public identity P_i and most messages will be transactions of the following form: $m = (P_i, P_j, \text{q}, \text{aux})$ where P_i transfers q coins to P_j along with some optional, auxiliary information aux.

4.4.2 Blockchain Setup and Key Knowledge.

As in [DGKR18], we assume that the genesis block is generated by an initialization functionality $\mathcal{F}_{\text{INIT}}$ that registers all parties' keys. Moreover, we assume that primitives specified in separate functionalities in [DGKR18] as incorporated into $\mathcal{F}_{\text{INIT}}$. $\mathcal{F}_{\text{INIT}}$ is executed by the environment $\mathcal{Z}$ as defined below and is parameterized by a stake distribution associating each party P_i to an initial stake stake_i. Upon being activated by P_i for the first time, $\mathcal{F}_{\text{INIT}}$ generates a signature key pair $\text{Sig.sk}_i, \text{Sig.pk}_i$, auxiliary information aux_i and a lottery witness $\text{sk}_{L,i}$, which will be defined as part of the lottery predicate in Section 4.4.4, sending $(\text{Sig.sk}_i, \text{Sig.pk}_i, \text{aux}_i, \text{sk}_{L,i}, \text{stake}_i)$ to P_i as response. After

all parties have activated $\mathcal{F}_{\mathsf{INIT}}$, it responds to requests for a genesis block by providing $B_0 = \{(\mathsf{Sig.pk}_1, \mathsf{aux}_1, \mathsf{stake}_1), \ldots, (\mathsf{Sig.pk}_n, \mathsf{aux}_n, \mathsf{stake}_n), \mathsf{aux}\}$, where aux is generated according to the underlying blockchain consensus protocol.

Since $\mathcal{F}_{\mathsf{INIT}}$ generates keys for all parties, we capture the fact that even corrupted parties have registered public keys and auxiliary information such that they know the corresponding secret keys. Moreover, when our EtF constructions are used as part of more complex protocols, a simulator executing the EtF and its underlying blockchain with the adversary will be able to predict which ciphertexts can be decrypted by the adversary by simulating $\mathcal{F}_{\mathsf{INIT}}$ and learning these keys. This fact will be important when arguing the security of protocols that use our notion of EtF.

4.4.3 Evolving Blockchains.

In the coming chapters we will consider the concept of *future* in a blockchain context. In particular we want to make sure that the initial chain $\mathbf{B}$ has "correctly" evolved into the final chain $\tilde{\mathbf{B}}$. Otherwise, the adversary can easily simulate a blockchain where it wins a future lottery. Fortunately, the *Distinguishable Forking* property provides just that (see Definition 4.6 and [GG17] for more details). A sufficiently long chain in an honest execution can be distinguished from a fork generated by the adversary by looking at the combined amount of stake proven in such a sequence of blocks. We encapsulate this property in a predicate called $\mathsf{evolved}(\cdot,\cdot)$. First, let $\Gamma^V = (\mathsf{UpdateState}^V, \mathsf{GetRecords}, \mathsf{Broadcast})$ be a blockchain protocol with validity predicate V and where the $(\alpha, \beta, \ell_1, \ell_2)$-*distinguishable forking* property holds. And let $\mathbf{B} \leftarrow \mathsf{GetRecords}(1^\lambda, \mathsf{st})$ and $\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(1^\lambda, \tilde{\mathsf{st}})$.

Definition 4.7 (Evolved Predicate). *An evolved predicate is a polynomial time function* evolved *that takes as input blockchains* $\mathbf{B}$ *and* $\tilde{\mathbf{B}}$

$$\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) \in \{0, 1\}$$

It outputs 1 iff $\mathbf{B} = \tilde{\mathbf{B}}$ *or the following holds (i)* $V(\mathbf{B}) = V(\tilde{\mathbf{B}}) = 1$*; (ii)* $\mathbf{B}$ *and* $\tilde{\mathbf{B}}$ *are consistent i.e.* $\mathbf{B}^{\lceil\kappa} \preceq \tilde{\mathbf{B}}$ *where* κ *is the common prefix parameter; (iii) Let* $\ell' = |\tilde{\mathbf{B}}| - |\mathbf{B}|$ *then it holds that* $\ell' \geq \ell_1 + \ell_2$ *and* $\mathsf{u\text{-}stakefrac}(\tilde{\mathbf{B}}, \ell' - \ell_1) > \beta$.

4.4.4 Blockchain Lotteries.

Earlier we mentioned the concept of leader election in PoS-based blockchain protocols. In this kind of lottery any party can win the right to become a slot leader with a probability proportional to its relative stake in the system. Usually, the lottery winner wins the right to propose a new block for the chain, introduce new randomness to the system or become a part of a committee that carries out some computation.

Independent Lotteries. In some applications it is useful to conduct multiple independent lotteries for the same slot sl. Therefore we associate each slot with a set of roles $\mathsf{R}_1, \ldots, \mathsf{R}_n$. Depending on the lottery mechanism, each pair $(\mathsf{sl}, \mathsf{R}_i)$ may yield zero, one or multiple winners. Often, a party can locally compute if it, in fact, is the lottery winner for a given role and the evaluation procedure may equip the party with a proof for others to verify. The below definition details what it means for a party to win a lottery.

Definition 4.8 (Lottery Predicate). *A lottery predicate is a polynomial time function* $\mathsf{lottery}$ *that takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* R *and a lottery witness* $\mathsf{sk}_{L,i}$ *and outputs 1 if and only if the party owning* $\mathsf{sk}_{L,i}$ *won the lottery for the role* R *in slot* sl *with respect to the blockchain* $\mathbf{B}$.
Formally, we write

$$\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) \in \{0, 1\}$$

It is natural to establish the set of lottery winning keys $\mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{R}}$ for parameters $(\mathbf{B}, \mathsf{sl}, \mathsf{R})$. This is the set of eligible keys satisfying the $\mathsf{lottery}$ predicate.

Chapter 5

Cryptographic Primitives

5.1 Secret Sharing

A central tool used for securely implementing basic primitives for layered MPC is secret sharing. A (t, n)-secure secret sharing scheme takes a secret s as input and computes n shares with the secrecy guarantee that any set of at most t shares reveal nothing about s while any set of $t + 1$ correct shares can be used to efficiently reconstruct the secret. A robust secret sharing scheme additionally guarantees correct reconstruction from n shares, even when an arbitrary set of t out of the n shares have been tampered. Formally,

Definition 5.1 (Secret Sharing). *A secret sharing scheme with secret domain S and share domain U is a pair of algorithms* $(\mathsf{Sh}, \mathsf{Rec})$ *called the sharing algorithm and the reconstruction algorithm, respectively.*

- ***Sharing.*** Sh *takes as input a secret $s \in S$ and randomness ρ chosen uniformly from some domain R and computes* $\mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n)$*, where $s_i \in U$ will be called the ith share of the secret s. If ρ is clear from the context we just write* $\mathsf{Sh}(s) = (s_1, \ldots, s_n)$.

- ***Reconstruction.*** Rec *takes the set of all shares and computes* $\mathsf{Rec}(s_1, \ldots, s_n) = \hat{s}$.

The (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ *should satisfy the following properties:*

Correctness. *For all $\rho \leftarrow R$, and any $s \in S$, we have* $\mathsf{Rec}(\mathsf{Sh}(s, \rho)) = s$.

t-security. *For any $s, s' \in S$, and $I \subset [n], |I| \leq t$,*

$$\left\{\{s_i\}_{i\in I} \middle| \rho \leftarrow R, \mathsf{Sh}(s, \rho) = \{s_i\}_{i\in[n]}\right\} \equiv \left\{\{s_i\}_{i\in I} \middle| \rho \leftarrow R, \mathsf{Sh}(s', \rho) = \{s_i\}_{i\in[n]}\right\}.$$

The scheme $(\mathsf{Sh}, \mathsf{Rec})$ *is said to be robust against t corruptions if it additionally satisfies the following reconstruction property:*

t-robustness. *For any $\rho \leftarrow R$, a secret $s \in S$, a set $I \subset [n]$ of size at most t, and shares $\hat{s}_i \in U$, for $i \in I$,*

$$\mathsf{Rec}(\{s_i\}_{i\in[n]\setminus I}, \{\hat{s}_i\}_{i\in I}) = s, \text{ where } \mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n).$$

We next define two commonly used, robust secret sharing schemes.

Definition 5.2 (Threshold CNF-Secret Sharing [ISN89])**.** *Let $t, n \in \mathbb{N}$ and s be an element in a ring $\mathbb{L}$. Let $\mathcal{T}$ be an enumeration of all $\binom{n}{n-t}$ subsets of size $n-t$ such that $\mathcal{T} = \{T \subseteq 2^{[n]} : |T| = n-t\} = \{T_1, \ldots, T_k\}$. A threshold CNF-secret sharing scheme* (Sh, Rec) *is defined as follows:*

- ***Sharing.*** *To share a secret s, first sample $\{r_T\}_{T\in\mathcal{T}}$ uniformly at random from $\mathbb{L}$ subject to $s = \sum_{T\in\mathcal{T}} r_T$ (i.e., obtain a k-out-of-k additive secret sharing of s). Let s_i consist of all shares r_T such that $T \in \mathcal{T}$ and $i \in T$. Then,* $\mathsf{Sh}(s, \rho) = (s_1, \ldots, s_n)$.

- ***Robust Reconstruction.*** *Let $\{s_i\}_{i\in[n]}$ be the set of input shares. For each $i \in [n]$ and $T \in \mathcal{T}$ such that $i \in T$, denote the additive share r_T from s_i as r_T^i. For each $T \in \mathcal{T}$, choose the unique value $\hat{r}_T$ such that $|\{i \in [n] : r_T^i = \hat{r}_T\}| \geq n - 2t$. Then,* $\mathsf{Rec}(\{s_i\}_{i\in[n]}) = \sum_{T\in\mathcal{T}} \hat{r}_T$.

Definition 5.3 (Shamir Secret Sharing)**.** *For t, n such that $t < n/3$, a (t, n)-Shamir secret sharing scheme* (Sh, Rec) *is defined using a finite field $\mathbb{F}$ such that $|\mathbb{F}| > n$ and distinct non-zero field elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$.*

- ***Sharing.*** *Given a secret $s \in \mathbb{F}$,* Sh *samples $c_1, \ldots, c_t$ uniformly and independently from $\mathbb{F}$ and defines the polynomial $p(x) = s + \sum_{i=1}^{t} c_i x^i$. Then,* $\mathsf{Sh}(s) = (s_1, \ldots, s_n)$*, where $s_i = p(\alpha_i)$.*

- ***Robust Reconstruction.*** *Let $\{\hat{s}_i\}_{i\in[n]}$ be the set of received shares.* Rec *finds a polynomial $p(x)$ of degree at most t such that $|\{i : p(\alpha_i) \neq \hat{s}_i\}| \leq t$. Then,* $\mathsf{Rec}(\{\hat{s}_i\}_{i\in[n]}) = p(0)$.

5.1.1 Shamir Secret Sharing on Groups of Order p

The well known degree-t Shamir scheme allows to split a secret $s \in \mathbb{Z}_p$ in n shares (where $0 \leq t < n < p$) in such a way that any set of $t+1$ shares give full information about the secret s while any set of t give no information on s.

Here we will consider situations where the secret is an element $S = sG$ of a group $\mathbb{G}$ of order p with generator G, but the dealer does not know s (and hence cannot apply the usual Shamir sharing using s as secret). On the other hand, it is enough that the shares allow to reconstruct S and not s. We define Shamir secret sharing in a group of order p as shown in Fig. 5.1 (Shamir secret sharing scheme over $\mathbb{Z}_p$ is retrieved by setting $\mathbb{G} = (\mathbb{Z}_p, +)$, $G = 1$). We denote by $\mathbb{Z}_p[X]_{\leq t}$ the set of polynomials in $\mathbb{Z}_p[X]$ of degree at most t.

Shamir sharing on a group of order p

PUBLIC PARAMETERS: Let $pp = (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\})$, where $\mathbb{G}$ is a group of prime order p with generator G, $0 \leq t < n < p$ are integers, and $\alpha_0, \alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p$ are pairwise distinct.

Algorithm 1: GShamir.Share(pp, S)

1: **Input:** $S \in \mathbb{G}$
2: $m(X) \leftarrow\$ \{m(X) \in \mathbb{Z}_p[X]_{\leq t} : m(\alpha_0) = 0\}$
3: $A_i = S + m(\alpha_i) \cdot G,\ i \in [n]$
4: **return** $(A_1, \ldots, A_n)$

Algorithm 2: GShamir.Rec($pp, I, \{A_i\}_{i \in I}$)

1: **Input:** $I \subseteq [n], |I| = t + 1, \{A_i\}_{i \in I}$
2: $\forall i \in I, \quad \lambda_{i,I} \leftarrow \prod_{j \in I, j \neq i} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j}$
3: $S' \leftarrow \sum_{i \in I} \lambda_{i,I} A_i$
4: **return** S'

Figure 5.1: Shamir secret sharing on a group $\mathbb{G}$ of order p

5.2 Commitment Schemes

We recall the syntax for a commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ below:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ outputs a commitment key. The commitment key ck defines a message space $\mathcal{S}_m$ and a randomizer space $\mathcal{S}_r$.
- $\mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho) \to \mathsf{cm}$ outputs a commitment given as input a message $\mathsf{s} \in \mathcal{S}_m$ and randomness $\rho \in \mathcal{S}_r$.

We require a commitment scheme to satisfy the standard properties of *binding* and *hiding*. It is binding if no efficient adversary can come up with two pairs $(\mathsf{s}, \rho), (\mathsf{s}', \rho')$ such that $\mathsf{s} \neq \mathsf{s}'$ and $\mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho) = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}'; \rho')$ for $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$. The scheme is hiding if for any two $\mathsf{s}, \mathsf{s}' \in \mathcal{S}_m$, no efficient adversary can distinguish between a commitment of s and one of s'.

Extractability. In our construction of ECW from cWE (Section 6.4.1), we require our commitments to satisfy an additional property which allows to *extract* message and randomness of a commitment. In particular we assume that our setup outputs both a commitment key and a trapdoor td and that there exists an algorithm Ext such that $\mathsf{Ext}(\mathsf{td}, \mathsf{cm})$ outputs (s, ρ) such that $\mathsf{cm} = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$. We remark we can generically obtain this property by attaching to the commitment a NIZK argument of knowledge that shows knowledge of opening, i.e., for the relation $\mathcal{R}^{\mathsf{opn}}(\mathsf{cm}_i; (\mathsf{s}, \rho)) \iff \mathsf{cm}_i = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$.

5.3 Oblivious Transfer.

A 2-round oblivious transfer (OT) protocol between a receiver R and a sender S consists of three polynomial-time algorithms $\Pi_{\mathsf{OT}} = (\Pi^R_{\mathsf{OT}}, \Pi^S_{\mathsf{OT}}, \Pi^O_{\mathsf{OT}})$:

$m^R \leftarrow \Pi^R_{\mathsf{OT}}(b; r^R)$**.** In the first round, the receiver R on input $b \in \{0,1\}$ and random tape $r^R \in \{0,1\}^{\mathsf{poly}(\lambda)}$ generates the OT first message m^R.

$m^S \leftarrow \Pi^S_{\mathsf{OT}}(m^R, (x^0, x^1); r^S)$**.** In the second round, the sender S on input (x^0, x^1), where $x^l \in \{0,1\}^{\mathsf{poly}(\lambda)}$ for $l \in \{0,1\}$, generates the second message m^S using random tape $r^S \in \{0,1\}^{\mathsf{poly}(\lambda)}$.

$x \leftarrow \Pi^O_{\mathsf{OT}}(m^S, b, r^R)$**.** R computes the output $x = \Pi^O_{\mathsf{OT}}(m^S, b, r^R)$.

We require an OT protocol to securely implements the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ given in Fig. 5.2 in the presence of malicious adversaries.

Functionality for OT

Choose. On input $(\texttt{receive}, \mathsf{sid}, b)$ from R , where $b \in \{0,1\}$, if no messages of the form $(\texttt{receive}, \mathsf{sid}, b)$ is stored, store $(\texttt{receive}, \mathsf{sid}, b)$ and send $(\texttt{receive}, \mathsf{sid})$ to S.

Transfer. On input $(\texttt{send}, \mathsf{sid}, x^0, x^1)$ from S, with $x^0, x^1 \in \{0,1\}^k$, if no messages of the form $(\texttt{send}, \mathsf{sid}, x^0, x^1)$ is stored and a message of the form $(\texttt{receive}, \mathsf{sid}, b)$ is present, send $(\texttt{sent}, \mathsf{sid}, x^b)$ to R.

Figure 5.2: The ideal functionality $\mathcal{F}_{\mathsf{OT}}$ for oblivious transfer

5.4 Circuit-related Primitives

5.4.1 Layered Circuits

Definition 5.4 (Layered Arithmetic Circuits)**.** *Let $\mathbb{F}$ be a field and $C : \mathbb{F}^n \to \mathbb{F}^m$ be an arithmetic circuit over $\mathbb{F}$ of size $|C| = M$ with gates $g_1, \ldots, g_M$ of the following types.*

- ***Addition:*** *Given input wire values $x_1, \ldots, x_k \in \mathbb{F}$, the output of the gate is $z = \sum_{i \in [k]} x_i$.*
- ***Multiplication-by-Constant:*** *Given input wire value $x \in \mathbb{F}$ and a constant $c \in \mathbb{F}$ associated with the gate, output the value $z = c \cdot x$.*
- ***Multiplication:*** *Given input wire values x_1 and x_2 from $\mathbb{F}$, the output wire value is the product $z = x_1 \cdot x_2$.*

C is a layered arithmetic circuit if the gates can be partitioned into layers $L_0, \ldots, L_D$ such

that for every layer $1 \leq i \leq D$ *and every gate* $g \in L_i$*, all inputs to* g *are outputs of gates in* L_{i-1}.

5.4.2 Garbled Circuits.

We recall the definition of *garbling schemes* formalized by Bellare et al. in [BHR12].

Definition 5.5 (Garbling Scheme). *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a polynomial-size circuit class. A garbled circuit scheme* GC *for* $\mathcal{C}$ *consists of four polynomial-time algorithms* $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$*:*

$(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C)$**:** *On input a boolean circuit* $C \in \mathcal{C}_\lambda$*, outputs* $(\mathbf{C}, e, d)$*, where* $\mathbf{C}$ *is a garbled circuit,* e *is encoding information, and* d *is decoding information.*

$X \leftarrow \mathsf{Encode}(e, x)$**:** *On input* e *and* x*, where* x *is a suitable input for* C*, outputs a garbled input* X.

$Y = \mathsf{Eval}(\mathbf{C}, X)$**:** *On input* $(\mathbf{C}, X)$ *as above, outputs a garbled output* Y.

$y \leftarrow \mathsf{Decode}(d, Y)$**:** *On input* (d, Y) *as above, outputs a plain output* y.

For our construction, we are interested in garbling schemes with the following properties.

Correctness. For any security parameter $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}_\lambda$, for $(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C)$, and for all suitable input x:

$$\mathsf{Decode}(d, \mathsf{Eval}(\mathbf{C}, \mathsf{Encode}(e, x))) = C(x)$$

Authenticity. For all circuits $C : \{0,1\}^n \to \{0,1\}$, inputs $x \in \{0,1\}^n$, where $n = \mathsf{poly}(\lambda)$, and for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\begin{matrix}\hat{Y} \neq \mathsf{Eval}(\mathbf{C}, X) \wedge \\ \mathsf{Decode}(d, \hat{Y}) \neq \bot\end{matrix} : \begin{matrix}(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C) \\ X = \mathsf{Encode}(e, x);\ \hat{Y} \leftarrow \mathcal{A}(C, x, \mathbf{C}, X)\end{matrix}\right] \geq 1 - \mathsf{negl}(\lambda)$$

5.5 (Threshold) Identity Based Encryption

We recall the definition of an identity-based encryption (IBE) scheme [BF01].

5.5.1 IBE

An IBE scheme Π_{IBE} consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$**.** The setup algorithm takes as input a security parameter λ and returns a master key msk together with some publicly known system parameters sp including a master public key mpk, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. We assume that all algorithms takes sp as input implicitly.

$\mathsf{IDKeygen}(\mathsf{msk}, \mathsf{ID})$**.** The identity key-generation algorithm takes as input msk and an identity $\mathsf{ID} \in \{0,1\}^*$, and returns a decryption key $\mathsf{sk}_{\mathsf{ID}}$ for ID.

$\mathsf{Enc}(\mathsf{ID}, m)$**.** The encryption algorithm takes as input an identity string $\mathsf{ID} \in \{0,1\}^*$ and $m \in \mathcal{M}$. It returns a ciphertext $ct \in \mathcal{C}$.

$\mathsf{Dec}(ct, \mathsf{sk}_{\mathsf{ID}})$**.** The decryption algorithm takes as input $ct \in \mathcal{C}$ and a decryption key $\mathsf{sk}_{\mathsf{ID}}$. It returns $m \in \mathcal{M}$.

Correctness. An IBE scheme Π_{IBE} should satisfy the standard correctness property, namely for $\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{IDKeygen}(\mathsf{msk}, \mathsf{ID})$ and for any $m \in \mathcal{M}$, we must have:

$$\mathsf{Dec}(\mathsf{Enc}(\mathsf{ID}, m), \mathsf{sk}_{\mathsf{ID}}) = m.$$

where $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$

Security. We use adaptive-identity security [BF01]. After the challenger runs the setup algorithm, the adversary has access to an oracle $\mathcal{O}_{\mathsf{msk}}$ that on input any id, returns sk_{id}. $\mathcal{A}$ may query the oracle on arbitrary identities of its choice even before selecting the messages m_0, m_1. More formally, we say that Π_{IBE} is secure if any PPT adversary $\mathcal{A}$ has only negligibly greater than 1/2 probability of correctly guessing the bit b in the following game:

1. The challenger runs Setup and outputs sp to $\mathcal{A}$.
2. $\mathcal{A}$ may query the oracle $\mathcal{O}_{\mathsf{msk}}$ that on any input id returns sk_{id}.
3. $\mathcal{A}$ outputs a target identity id^* and two equal-size messages $m_0, m_1 \in \mathcal{M}$.
4. The challenger selects a random bit b and outputs $c^* \leftarrow \mathsf{Enc}(id^*, m_b)$ to $\mathcal{A}$.
5. $\mathcal{A}$ may continue to query $\mathcal{O}_{\mathsf{msk}}$ on any input $id \neq id^*$.
6. $\mathcal{A}$ outputs b'.

where $\mathcal{O}_{\mathsf{msk}}(\mathsf{ID})$ outputs $\mathsf{IDKeygen}(\mathsf{msk}, \mathsf{ID})$.

5.5.2 Threshold IBE.

A TIBE system consists of the following algorithms.

$\Pi_{\mathsf{TIBE}}.\mathsf{Setup}(1^\lambda, n, k) \rightarrow (\mathsf{sp}, \mathsf{vk}, \vec{\mathsf{msk}})$: It outputs some public system parameters sp (including mpk), verification key vk, and vector of master secret key shares $\vec{\mathsf{msk}} =$

$(\mathsf{msk}_1, \ldots, \mathsf{msk}_n)$ for n with threshold k. We assume that all algorithms takes sp as input implicitly.

$\Pi_{\mathsf{TIBE}}.\mathsf{ShareKG}(i, \mathsf{msk}_i, \mathsf{ID}) \to \theta = (i, \hat{\theta})$: It outputs a private key share $\theta = (i, \hat{\theta})$ for ID given a share of the master secret key.

$\Pi_{\mathsf{TIBE}}.\mathsf{ShareVerify}(\mathsf{vk}, \mathsf{ID}, \theta) \to 0/1$: It takes as input the verification key vk, an identity ID, and a share of master secret key θ, and outputs 0 or 1.

$\Pi_{\mathsf{TIBE}}.\mathsf{Combine}(\mathsf{vk}, \mathsf{ID}, \vec{\theta}) \to \mathsf{sk}_{\mathsf{ID}}$: It combines the shares $\vec{\theta} = (\theta_1, \ldots, \theta_k)$ to produce a private key $\mathsf{sk}_{\mathsf{ID}}$ or $\perp$.

$\Pi_{\mathsf{TIBE}}.\mathsf{Enc}(\mathsf{ID}, m) \to \mathsf{ct}$: It encrypts message m for identity ID and outputs a ciphertext ct.

$\Pi_{\mathsf{TIBE}}.\mathsf{Dec}(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ct}) \to m$: It decrypts the ciphertext ct given a private key $\mathsf{sk}_{\mathsf{ID}}$ for identity ID.

Correctness. A TIBE scheme Π_{TIBE} should satisfy two correctness properties:

1. For any identity ID, if $\theta = \Pi_{\mathsf{TIBE}}.\mathsf{ShareKG}(i, \mathsf{msk}_i, \mathsf{ID})$ for $\mathsf{msk}_i \in \vec{\mathsf{msk}}$, then $\Pi_{\mathsf{TIBE}}.\mathsf{ShareVerify}(\mathsf{vk}, \mathsf{ID}, \theta) = 1$.

2. For any ID, if $\vec{\theta} = \{\theta_1, \ldots, \theta_k\}$ where $\theta_i = \Pi_{\mathsf{TIBE}}.\mathsf{ShareKG}(i, \mathsf{msk}_i, \mathsf{ID})$, and $\mathsf{sk}_{\mathsf{ID}} = \Pi_{\mathsf{TIBE}}.\mathsf{Combine}(\mathsf{vk}, \mathsf{ID}, \vec{\theta})$, then for any $m \in \mathcal{M}$ and $\mathsf{ct} = \Pi_{\mathsf{TIBE}}.\mathsf{Enc}(\mathsf{ID}, m)$ we have $\Pi_{\mathsf{TIBE}}.\mathsf{Dec}(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ct}) = m$.

Structural Property: TIBE as IBE + Secret Sharing. We model threshold IBE in a modular manner from IBE and assume it to have a certain structural property: that it can be described as an IBE "lifted" through a homomorphic secret-sharing [BGI+18, BBH06, Nie03]. TIBE constructions can often be described as such. We assume this structural property to later be able to construct proofs modularly, but we remark our our results do not depend on it and they hold for an arbitrary TIBE.

5.5.3 Constructing TIBE from IBE and Homomorphic Secret Sharing.

Assume a secure $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{IDKeygen}, \mathsf{Enc}, \mathsf{Dec})$. We can transform it into a threshold IBE using homomorphic secret sharing algorithms $(\mathsf{Share}, \mathsf{EvalShare}, \mathsf{Combine})$. A homomorphic secret sharing scheme is a secret sharing scheme with an extra property: given a shared secret, it allows to compute a share of a function of the secret on it. It has the following syntax (which we specialize for the IBE setting):

- $\mathsf{Share}(\mathsf{msk}, k, n) \to (\mathsf{msk}_1, \ldots, \mathsf{msk}_n)$ shares the secret.

- $\mathsf{EvalShare}(\mathsf{msk}_i, f) \to y_i$ obtains a share for $f(\mathsf{msk})$ where f is a function.

- $\mathsf{Combine}((y_i)_{i\in T}) \to y^*$ where T is a set with size above threshold.

We assume all the algorithms above take as input the master public-key for simplicity. The correctness of the homomorphic scheme requires that running $y_i \leftarrow \mathsf{EvalShare}(\mathsf{msk}_i, f)$ on msk_i output of Share and then running $\mathsf{Combine}$ on (a large enough set of) the y_i-s produces the same output as $f(\mathsf{msk})$. We also require that $\mathsf{Combine}$ can reconstruct msk from a large enough set of the msk_i-s.

The construction for threshold IBE is now straightforward:

- at setup time, we produce shares $\mathsf{msk}_1, \ldots, \mathsf{msk}_n$ of the master secret key using the Share algorithm on the master secret key output of Setup.
- encryption is syntactically and functionally the same in both cases.
- to produce a partial secret-key for a certain id, we just run $\mathsf{sk}_i^{\mathsf{ID}} \leftarrow \mathsf{EvalShare}(\mathsf{msk}_i, \mathsf{IBE.IDKeygen}(\mathsf{mpk}, \cdot, \mathsf{ID}))$.
- for decryption, given enough shares for an ID ID, we run on them algorithm $\mathsf{Combine}$ to obtain $\mathsf{sk}_{\mathsf{ID}}$; we then simply run $\mathsf{IBE.Dec}$.

Threshold IBE security. If the homomorphic secret sharing supports up to a threshold k, then we obtain analogous properties for the threshold IBE construction. In particular the threshold IBE satisfies the following simulation properties for any n and threshold k supported by the homomorphic secret sharing scheme[1].

Master secret-key share simulation. For any PPT adversary $\mathcal{A}$ there exists a simulator $\mathsf{Sim}_{\mathsf{msk}}$ such that the following two distributions are indistinguishable.

$$\begin{aligned}
&\{(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}}) : (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda);\\
&\qquad\qquad S_{\mathsf{corr}} \leftarrow \mathcal{A}(\mathsf{mpk}); (\mathsf{msk}_i)_{i\in n} \leftarrow \mathsf{Share}(\mathsf{msk}, n, k)\} \approx\\
&\{(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}}) : (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda);\\
&\qquad\qquad S_{\mathsf{corr}} \leftarrow \mathcal{A}(\mathsf{mpk}); (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}} \leftarrow \mathsf{Sim}_{\mathsf{msk}}(\mathsf{mpk}, S_{\mathsf{corr}}, n, k)\}
\end{aligned}$$

Key-generation simulation. For any PPT adversary there exists a simulator $\mathsf{Sim}_{\mathsf{kg}}$ such that the following two distributions are indistinguishable.

$$\begin{aligned}
\{(\mathsf{mpk}, (\mathsf{sk}_i^{\mathsf{ID}})_{i\in[n]}) : &(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda);\\
&S_{\mathsf{corr}} \leftarrow \mathcal{A}(\mathsf{mpk}); (\mathsf{msk}_i)_{i\in n} \leftarrow \mathsf{Share}(\mathsf{msk}, n, k);\\
&\mathsf{ID} \leftarrow \mathcal{A}(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}});
\end{aligned}$$

[1] The security of this type of construction is proven for example in [Nie03] to which we defer the reader for details.

$$\begin{aligned}
&\mathsf{sk}_i^{\mathsf{ID}} \leftarrow \mathsf{EvalShare}(\mathsf{msk}_i, \mathsf{ID}) \text{ for } i \in [n]\} \approx \\
\{(\mathsf{mpk}, (\mathsf{sk}_i^{\mathsf{ID}})_{i\in[n]}) :&(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda); \\
&S_{\mathsf{corr}} \leftarrow \mathcal{A}(\mathsf{mpk}); (\mathsf{msk}_i)_{i\in n} \leftarrow \mathsf{Share}(\mathsf{msk}, n, k); \\
&\mathsf{ID} \leftarrow \mathcal{A}(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}}); \\
&(\mathsf{sk}_i^{\mathsf{ID}})_{i\in[n]} \leftarrow \mathsf{Sim}_{\mathsf{kg}}(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\mathsf{corr}}}, \mathsf{ID})\}
\end{aligned}$$

Robustness of TIBE. We assume a *robust* threshold IBE scheme, where we can verify that each of the ID-specific shares are authenticated, i.e. they have been produced by a party with the related master secret key share. This property can be obtained by assuming an underlying secret sharing scheme which is itself robust. This in turn can be obtained by attaching a NIZK or a homomorphic signature to the share.

TIBE with Proactive Secret Sharing. We assume our TIBE to allow for the shares of the master secret keys to be reshared among the committee members which evolve through time. With this goal in mind we can consider a proactive secret sharing scheme which includes a *handover* (each committee member can reshare its share) and *reconstruction* stage (committee members in a new epoch can reconstruct their secret from the output of the handover). We can directly extend a TIBE with such syntax. The resulting scheme should provide the same simulation properties as the ones described above for the non proactive case.

5.6 Smooth Projective Hash Function (SPHF)

Let $\mathcal{L}_{\mathtt{lpar}}$ be a NP language, parametrized by a language parameter $\mathtt{lpar}$, and $\mathcal{R}_{\mathtt{lpar}} \subseteq \mathcal{X}_{\mathtt{lpar}}$ be its corresponding relation. A Smooth projective hash functions (SPHFs, [CS02]) for $\mathcal{L}_{\mathtt{lpar}}$ is a cryptographic primitive with this property that given $\mathtt{lpar}$ and a statement $\mathtt{x}$, one can compute a hash of $\mathtt{x}$ in two different ways: either by using a projection key hp and $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathtt{lpar}}$ as $\mathsf{pH} \leftarrow \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$, or by using a hashing key hk and $\mathtt{x} \in \mathcal{X}_{\mathtt{lpar}}$ as $\mathsf{H} \leftarrow \mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$.

Definition 5.6. *A SPHF for $\{\mathcal{L}_{\mathtt{lpar}}\}_{\mathtt{lpar}}$ is a tuple of PPT algorithms* $(\mathsf{setup}, \mathsf{hashkg}, \mathsf{projkg}, \mathsf{hash}, \mathsf{projhash})$*, which are defined as follows:*

$\mathsf{setup}(1^\lambda)$**:** *Takes in a security parameter λ and generates the global parameters $\mathtt{p}$ together with the language parameters $\mathtt{lpar}$. We assume that all algorithms have access to $\mathtt{p}$.*

$\mathsf{hashkg}(\mathtt{lpar})$**:** *Takes in a language parameter $\mathtt{lpar}$ and outputs a hashing key hk.*

$\mathsf{projkg}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$**:** *Takes in a hashing key hk, $\mathtt{lpar}$, and a statement $\mathtt{x}$ and outputs a projection key hp, possibly depending on $\mathtt{x}$.*

$\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$**:** *Takes in a hashing key* hk*,* $\mathtt{lpar}$*, and a statement* $\mathtt{x}$ *and outputs a hash value* H.

$\mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$**:** *Takes in a projection key* hp*,* $\mathtt{lpar}$*, a statement* $\mathtt{x}$*, and a witness* $\mathtt{w}$ *for* $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$ *and outputs a hash value* pH.

A SPHF has to fulfill two properties:

Correctness. For all $\mathtt{x} \in \mathcal{L}_{\mathtt{lpar}}$ and their corresponding witnesses $\mathtt{w}$, we have that $\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x}) = \mathsf{projhash}(\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w})$.

Smoothness. For any $\mathtt{lpar}$ and any $\mathtt{x} \notin \mathcal{L}_{\mathtt{lpar}}$, the hash value $\mathsf{hash}(\mathtt{lpar}; \mathsf{hk}, \mathtt{x})$ is indistinguishable from a random element in the set of hash values.

5.7 Sigma-protocols

Going forward, we will require non-interactive zero knowledge arguments of knowledge where most of our statements are instances of a general structure where we want to prove knowledge of preimage of some element via a *vector-space homomorphism* f: that is, let $\mathbb{F}$ be a finite field, $\mathcal{W}$ and $\mathcal{X}$ be $\mathbb{F}$-vector spaces, and $f : \mathcal{W} \to \mathcal{X}$ be a vector space homomorphism. Let

$$R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}.$$

The standard (Schnorr-like) Σ-protocol Π_{Pre} for this relation is as in Fig. 5.3.

5.7.1 Security of Π_{Pre}

The Σ-protocol Π_{Pre} is obviously complete. It has special soundness because given two accepting transcripts (a, e, z), (a, e', z') with $e \neq e'$, one can extract w as $(e - e')^{-1}(z - z')$. It is therefore a proof of knowledge of w with soundness error $1/|\mathbb{F}|$. Finally it is honest-verifier zero-knowledge: a simulator can produce a transcript that is indistinguishable from a real one by choosing z uniformly at random in $\mathcal{X}$, and e uniformly at random in $\mathbb{F}$, and then computing $a = f(z) - e \cdot x$, which is uniformly random in $\mathcal{W}$.

Generic Σ-protocol $\Pi_{\mathsf{Pre}}(w; x, f)$

Proof of knowledge of witness w for x with respect to the relation $R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}$.
PUBLIC PARAMETERS: Finite field $\mathbb{F}$, vector spaces $\mathcal{W}, \mathcal{X}$ over $\mathbb{F}$, vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$, $x \in \mathcal{X}$.
PROTOCOL:

1. The prover samples $r \leftarrow\$ \mathcal{W}$, sends $a = f(r)$ to the verifier.
2. The verifier samples $e \leftarrow\$ \mathbb{F}$, sends it to the sender.
3. The prover sends $z \leftarrow r + e \cdot w$ to the verifier.
4. The verifier accepts if $z \in \mathcal{W}$ and $f(z) = a + e \cdot x$.

Figure 5.3: Generic Σ-protocol for knowledge of homomorphism-preimage

A non-interactive zero-knowledge (NIZK) proof of knowledge in the random oracle model is obtained by applying the Fiat-Shamir transform (Fig. 5.4).

Generic non-interactive argument of knowledge $\Pi_{\mathsf{NI-Pre}}(w; x, f)$

Non-interactive argument of knowledge of witness for x for the relation $R_{\mathsf{Pre}} = \{(w, x) \in \mathcal{W} \times \mathcal{X} : x = f(w)\}$ in the random oracle model.
PUBLIC PARAMETERS: Finite field $\mathbb{F}$, vector spaces $\mathcal{W}, \mathcal{X}$ over $\mathbb{F}$, vector space homomorphism $f : \mathcal{W} \to \mathcal{X}$, $x \in \mathcal{X}$, random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$. Let $pp = (\mathbb{F}, \mathcal{W}, \mathcal{X}, \mathcal{H})$.

Algorithm 3: $\Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$

$r \leftarrow\$ \mathcal{W}$
$a \leftarrow f(r)$, $e \leftarrow \mathcal{H}(x, a)$, $z \leftarrow r + e \cdot w$
return $\pi \leftarrow (e, z)$

Algorithm 4: $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp, x, f, \pi)$

Parse $\pi = (e, z)$
return accept if and only if $z \in \mathcal{W}$ and $e = \mathcal{H}(x, f(z) - e \cdot x)$

Figure 5.4: Generic non-interactive argument of knowledge of homomorphism-preimage

5.7.2 Group Homomorphism Preimage, DL and DLEQ Knowledge Proofs

Some useful examples of homomorphism-preimage relations R_{Pre} are given by discrete logarithm and discrete logarithm equality. Indeed, a cyclic group $\mathbb{G}$ of prime order p has a vector space structure over the field $\mathbb{Z}_p$, and a group homomorphism $f : \mathbb{G} \to \mathbb{G}'$ between groups of order p is also a $\mathbb{Z}_p$-vector homomorphism.[2] Let G be a generator of $\mathbb{G}$. Given $X \in \mathbb{G}$, a discrete logarithm DL proof of knowledge $\mathsf{DL}(w; G, X)$ asserts knowledge of $w \in \mathbb{Z}_p$ with $X = w \cdot G$ (we denote this as $w = \mathsf{DL}_G(X)$). In the language above this is provided by $\Pi_{\mathsf{NI-Pre}}(w; (X), f_G)$ with $f_G(w) = w \cdot G$. This is the non-interactive version of the well known Schnorr proof.

[2] This extends to direct products of groups of order p, i.e. $\mathcal{W} = \mathbb{G}_1 \times \cdots \times \mathbb{G}_m$, $\mathcal{X} = \mathbb{G}'_1 \times \cdots \times \mathbb{G}'_n$ and $f = (f_1, \ldots, f_m) : \mathcal{W} \to \mathcal{X}$ where $f_i : \mathbb{G}_i \to \mathcal{X}$ are all group homomorphisms.

Similarly, let G, H be elements in $\mathbb{G}$. Given $X, Y \in \mathbb{G}$ the discrete logarithm equality proof $\mathsf{DLEQ}(w; G, X, H, Y)$ is a non-interactive proof of knowledge of $w \in \mathbb{Z}_p$ with $w = \mathsf{DL}_G(X) = \mathsf{DL}_H(Y)$, which can be obtained by using $\Pi_{\mathsf{NI-Pre}}(w; (X, Y), f_{(G,H)})$, where $f_{G,H}(w) := (w \cdot G, w \cdot H)$.

5.8 Basic Notions on Public Key Encryption

In this section we introduce well-known concepts on public key encryption.

5.8.1 Definitions

Definition 5.7. *A public key encryption scheme $\mathcal{E}$ consists of three polynomial time algorithms $(\mathcal{E}.g, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ as follows:*

- $\mathcal{E}.g(\lambda)$ *is a probabilistic algorithm that outputs a pair* $(\mathsf{sk}, \mathsf{pk})$ *consisting of a secret key and a public key.*
- $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ *is a probabilistic algorithm that takes as input a public key* pk *and a plaintext message M in a plaintext message space $\mathfrak{P}$ and outputs a ciphertexts C in a ciphertext space $\mathfrak{C}$. In addition, by abuse of notation, we define the function $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M; \rho)$ that specifies the result of $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ when randomness ρ (in a randomness space $\mathfrak{R}$) is used.*
- $\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(C)$ *is a deterministic function that takes secret key* sk*, and a ciphertext $C \in \mathfrak{C}$ and outputs a plaintext message $M' \in \mathfrak{P}$.*

and which satisfy that for every $(\mathsf{pk}, \mathsf{sk})$ *output by $\mathcal{E}.g$, and for every $M \in \mathfrak{P}$,*

$$\Pr[\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(\mathcal{E}.\mathsf{Enc}_{pk}(M))] = 1$$

The most well known notion of security for a public key encryption scheme is IND-CPA security, which requires that the encryptions of two messages under any public key pk are computationally indistinguishable without the knowledge of the corresponding sk. Here we consider the notion of ℓ-multi-key IND-CPA security. This requires that the encryptions of two vectors of messages of the same length, where each coordinate is encrypted under a public key pk_i, are indistinguishable. The notions are equivalent as long as ℓ is polynomial in the security parameter.

Definition 5.8. *A public key encryption scheme $\mathcal{E}$ satisfies ℓ-multi-key IND-CPA security if for any PPT adversary $\mathcal{B}$, there exists a negliglible function $\mu(\lambda)$ such that*

$$\left|\Pr\left[\mathsf{Game}^{\ell\text{-IND-CPA},0}_{\mathcal{B},\mathcal{E}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\ell\text{-IND-CPA},1}_{\mathcal{B},\mathcal{E}}(\lambda) = 1\right]\right| \leq \mu(\lambda)$$

Algorithm 5: $\mathsf{Game}^{\ell\text{-IND-CPA},b}_{\mathcal{B},\mathcal{E}}(\lambda)$

1: $\forall i \in [\ell] \quad (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathcal{E}.\mathsf{KeyGen}(pp, i)$
2: $(m_1^{(0)}, \dots, m_\ell^{(0)}), (m_1^{(1)}, \dots, m_\ell^{(1)}) \in \mathfrak{P}^\ell \leftarrow \mathcal{B}(pp, \{\mathsf{pk}_i : i \in [\ell]\})$
3: $\forall i \in [\ell], c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}_i}(m_i^{(b)})$
4: $b' \leftarrow \mathcal{B}(\{c_i : i \in [\ell]\})$
5: **return** b'

The case $\ell = 1$ is the usual IND-CPA definition and for $\ell = \mathsf{poly}(\lambda)$ a standard hybrid argument shows that a scheme is ℓ-multi-key IND-CPA if and only if it is IND-CPA.

5.8.2 El Gamal Public Key Encryption Scheme

We consider the well known El Gamal scheme, where the plaintext space is $\mathfrak{P} = \mathbb{G}$, a cyclic group of order p generated by G, the randomness space is $\mathfrak{R} = \mathbb{Z}_p$ and the ciphertext space is $\mathfrak{C} = \mathbb{G}^2$. The scheme $\mathcal{E}$ is given by

- $\mathcal{E}.g(\lambda)$: Selects $\mathsf{sk} \in \mathbb{Z}_p$ uniformly at random, sets $\mathsf{pk} = \mathsf{sk} \cdot G$, outputs $(\mathsf{sk}, \mathsf{pk})$.
- $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M)$ where $M \in \mathbb{G}$, selects $\rho \in \mathbb{Z}_p$ uniformly at random, outputs $C = (\rho \cdot G, M + \rho \cdot \mathsf{pk})$ (as explained before we denote $C = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(M; \rho)$).
- $\mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(C)$, where $C = (C_1, C_2) \in \mathbb{G}^2$, outputs $\mathsf{Dec}_{\mathsf{sk}}(C) = C_2 - \mathsf{sk} \cdot C_1$.

The El Gamal encryption scheme is well known to be IND-CPA secure under the DDH assumption.

5.8.3 $\mathbb{Z}_p$-linear Homomorphic Encryption

We now define encryption schemes with certain homomorphic properties, that allow for simple proofs of plaintext knowledge. These properties are attained by El Gamal encryption scheme (described in Section 5.8.2).

Definition 5.9 ($\mathbb{Z}_p$-linearly homomorphic encryption scheme). *Let $\mathcal{E} = (\mathcal{E}.g, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ be a public key encryption scheme and let p be a prime number. We say $\mathcal{E}$ is $\mathbb{Z}_p$-linearly homomorphic ($\mathbb{Z}_p$-LHE) if the plaintext space $(\mathfrak{P}, \boxplus_{\mathfrak{P}})$, randomness space $(\mathfrak{R}, \boxplus_{\mathfrak{R}})$, ciphertext space $(\mathfrak{C}, \boxplus_{\mathfrak{C}})$ each have a $\mathbb{Z}_p$-vector space structure and for all public keys pk output by $\mathcal{E}.g$, $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}} : \mathfrak{P} \times \mathfrak{R} \to \mathfrak{C}$ is a $\mathbb{Z}_p$-vector space homomorphism, i.e. for all $m_1, m_2 \in \mathfrak{C}$, $\rho_1, \rho_2 \in \mathfrak{R}$,*

$$\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_1; \rho_1) \boxplus_{\mathfrak{C}} \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_2; \rho_2) = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m_1 \boxplus_{\mathfrak{P}} m_2; \rho_1 \boxplus_{\mathfrak{R}} \rho_2).$$

Remark 5.1. *$\mathbb{Z}_p$-linear homomorphic encryption schemes have simple (non-interactive) proofs of plaintext (and randomness) knowledge, given by Fig. 5.4. More concretely, with notation as in that Figure, we take $\mathcal{W} = \mathfrak{P} \times \mathfrak{R}$, $\mathcal{X} = \mathfrak{C}$ and the proof $\Pi_{\mathsf{NI-Pre}}((m,\rho); c, \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}})$ for the relation $R_{\mathsf{Enc}} = \{((m,\rho), c) \in \mathcal{W} \times \mathcal{X} : c = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}(m;\rho)\}$.*

5.8.4 Proofs of Decryption Correctness

We will also need proofs of decryption correctness, where of course the prover wants to keep their secret key hidden, i.e. proofs for the relation

$$R_{\mathcal{E},\mathsf{Dec}} = \{(\mathsf{sk}; (\mathsf{pk}, m, c)) : (\mathsf{pk}, \mathsf{sk}) \text{ is a valid key-pair for } \mathcal{E} \text{ and } m = \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(c)\}$$

If the prover knows the randomness under which the message was encrypted, the proving algorithm $\mathcal{E}.\mathsf{ProveDec}(\mathsf{sk}; (\mathsf{pk}, m, c))$ can simply output that randomness $\pi \in \mathfrak{R}$; the verification $\mathcal{E}.\mathsf{VerifyDec}(\mathsf{pk}, m, c, \pi)$ accepts if $\mathsf{Enc}_{\mathsf{pk}}(m;\pi) = c$.

Unfortunately El Gamal encryption scheme does not allow a decryptor to retrieve the randomness under which a message has been encrypted. Instead, a proof of correctness of decryption for El Gamal can be constructed from the following property of this scheme, which we call $\mathbb{Z}_p$-linear decryption.

Definition 5.10. *Let $\mathcal{E} = (g, \mathsf{Enc}, \mathsf{Dec})$ be a $\mathbb{Z}_p$-linearly homomorphic encryption scheme and denote $\mathcal{PK}$ and $\mathcal{SK}$ the sets of public and secret keys respectively. $\mathcal{E}$ has $\mathbb{Z}_p$-linear decryption if:*

- *$\mathcal{PK}$ and $\mathcal{SK}$ are $\mathbb{Z}_p$-vector spaces.*
- *There exists a $\mathbb{Z}_p$-linear homomorphism $F : \mathcal{SK} \to \mathcal{PK}$ such that $\mathsf{pk} = F(\mathsf{sk})$ for all $(\mathsf{pk}, \mathsf{sk})$ outputted by g.*
- *For all $c \in \mathfrak{C}$, the function $D_c(\mathsf{sk}) := \mathsf{Dec}_{\mathsf{sk}}(c)$ is $\mathbb{Z}_p$-linear in sk, i.e. for all $\mathsf{sk}_1, \mathsf{sk}_2 \in \mathcal{SK}$, it holds that $D_c(\mathsf{sk}_1 \boxplus_{\mathcal{SK}} \mathsf{sk}_2) = D_c(\mathsf{sk}_1) \boxplus_{\mathfrak{P}} D_c(\mathsf{sk}_2)$.*

In this case we have the algorithms $(\mathcal{E}.\mathsf{ProveDec}, \mathcal{E}.\mathsf{VerifyDec})$ that constitute a NIZK proof for $R_{\mathcal{E},\mathsf{Dec}}$:

Algorithm 6: $\mathcal{E}.\mathsf{ProveDec}(\mathsf{sk}, (\mathsf{pk}, m, c))$

$\mathcal{W} \leftarrow \mathcal{SK}, \mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{P} \times \mathfrak{C}$,
$pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$w \leftarrow \mathsf{sk}$, $x \leftarrow (\mathsf{pk}, m)$,
$f(\cdot) \leftarrow (F(\cdot), D_c(\cdot))$
return
$\Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$

Algorithm 7: $\mathcal{E}.\mathsf{VerifyDec}(\mathsf{pk}, m, c, \pi)$

$\mathcal{W} \leftarrow \mathcal{SK}, \mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{P} \times \mathfrak{C}$
$pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
$x \leftarrow (\mathsf{pk}, m)$,
$f(\cdot) \leftarrow (F(\cdot), D_c(\cdot))$
return
$\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp, x, f)$

The El Gamal decryption function as usually described is not linear but affine, but we can easily fix this by e.g. defining $\mathsf{sk}^* = (\mathsf{sk}_1^*, \mathsf{sk}_2^*) = (1, \mathsf{sk}) \in \mathbb{Z}_p^2$ and letting $\mathsf{Dec}_{\mathsf{sk}^*}(C_1, C_2) := C_2 \cdot \mathsf{sk}_1^* - C_1 \cdot \mathsf{sk}_2^*$. Then $D_C(\mathsf{sk}^*)$ is clearly a $\mathbb{Z}_p$-linear function.

5.9 The SCRAPE Test

In SCRAPE [CD17], a technique for checking correctness of Shamir sharing in publicly verifiable secret sharing was introduced. Letting aside the details on how the technique works there, we are interested in the following fact, which in turn comes from well known results in coding theory [3].

Theorem 5.1 (SCRAPE dual-code test). *Let $1 \leq t < n$ be integers. Let p be a prime number with $p \geq n$. Let $\alpha_1, \dots, \alpha_n$ be pairwise different points in $\mathbb{Z}_p$. Define the coefficients $v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}$. Let*

$$C = \{(m(\alpha_1), \dots, m(\alpha_n)) : \ m(X) \in \mathbb{Z}_p[X]_{\leq t}\}.$$

Then, for every vector $(\sigma_1, \dots, \sigma_n)$ in $\mathbb{Z}_p^n$:

$$(\sigma_1, \dots, \sigma_n) \in C \quad \Leftrightarrow \quad \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot \sigma_i = 0, \quad \forall m^* \in \mathbb{Z}_p[X]_{\leq n-t-1}.$$

5.10 Mix Networks (Mixnets)

We use mixnets to anonymize a set of public encryption keys, each generated (with their corresponding secret keys) by a party in the system. Let $\mathcal{P}$ be the set of all parties generating these keys. We will assume such a mixnet and that the output is subsequently written to a blockchain. The output is a set of shuffled keys $\mathsf{pk}_{\mathsf{Anon},j} : j \in [n]$, for which each party knows the index that corresponds to their own public key, but nothing else about the permutation. Denote this permutation $\psi : \mathcal{P} \to [n]$, i.e. party $\mathcal{P}_i$ knows $j = \psi(i)$ and the corresponding key-pair.
We will use the fact that a party can encrypt a message under the public key $\mathsf{pk}_{\mathsf{Anon},j}$. It is clear that party $\mathcal{P}_{\psi^{-1}(j)}$ can decrypt the message, while the rest of the parties (even the sender) remain oblivious about the identity of the receiver. Notice that this setup can be instantiated via a verifiable mixnet (*e.g.* [BKRS18]).

[3] Specifically from the fact that the dual of a Reed-Solomon code is a generalized Reed-Solomon code of a certain form.

Part III

Publications

Chapter 6

Encryption to the Future [CDK+22]

6.1 Overview

Encryption to the Future (EtF)—Section 6.2. As in previous works [BGG+20, GHK+21, GHM+21], an EtF scheme is defined with respect to an underlying PoS blockchain. We naturally use core features of the PoS setting to define what "future" means. The vast majority of PoS blockchains (*e.g.* [DGKR18]) associates a *slot number* to each block and uses a lottery for selecting parties to generate blocks according to a stake distribution (*i.e.* the probability a party is selected is proportional to the stake the party controls). Thus, in EtF, we let a message be encrypted towards a party that is selected by the underlying blockchain's lottery scheme at a given future slot. We can generalize this and let the lottery select parties for multiple roles associated to each slot (so that committees consisting of multiple parties can be elected at a single point in time). We note that the goal of defining EtF with respect to an underlying blockchain is to construct it without having to assume very strong primitives such as (extractable) witness encryption for NP[1]. Moreover, it is necessary to provide a non-interactive EtF scheme with a means to publicly verify whether a given party has won the lottery to perform a certain role. Since this lottery predicate's output must hold for all parties, we need a consensus mechanism that allows for all parties to agree on lottery parameters/outputs while allowing for third parties to verify this result. An important point of our EtF definition is that it does not impose any constraints on the underlying blockchain's lottery scheme (*e.g.* it is not required to be anonymous) or on the slot when a party is supposed to be chosen to receive a message sent to a given role (*i.e.* party selection for a given role may happen w.r.t. a *future* stake distribution).

Relation to "Blockchain Witness Encryption" (BWE)—Section 6.7. In order to study how hard it is to realize EtF, we show that EtF implies a version of witness encryption [GGSW13] over a blockchain (similar to that of [GKM+22] but without relying on committees). The crux of the proof: if we can encrypt a message towards a role assigned to a party only at an arbitrary point in the future, then we can easily construct a witness encryption scheme exploiting EtF and a smart contract on the EtF's underlying blockchain. We also prove the opposite direction (BWE implies EtF), showing that the notions are similar from a feasibility standpoint. This shows another crucial point: to implement non-interactive EtF, we

[1] While one *might* define EtF in more general settings, namely without a blockchain, it is unclear how to obtain *interesting* instantiations, that is from standard primitives.

would plausibly need strong assumptions (e.g., full-blown WE). This follows by observing that existing constructions of WE over blockchains (e.g., [GKM+22]) are interactive in the sense that they rely on a committee that holds all encrypted messages in secret shared form and periodically re-share them. On the other hand, in the interactive setting, we show a construction of EtF with improved communication complexity that is *independent* from the size (or amount) of EtF encrypted messages: the committee only needs to hold an IBE master secret key (secret shared) and compute secret keys for specific identities. We note that the goal of constructing BWE from EtF is not to provide a concrete instantiation based on existing blockchains but rather to provide evidence that EtF is hard to construct from standard assumptions. The underlying blockchain protocol and lottery we use are standard Proof-of-Stake based blockchains with a VRF-based lottery and smart contracts. The only non-realistic assumption we make is that the stake is distributed in arbitrarily (i.e. it is all locked inside one smart contract) which is an assumption on how the blockchain is operated rather than on how it is constructed or why it is secure.

Encryption to the Current Winner (ECW)—Section 6.2. By the previous result we know that, unless we turn to strong assumptions, we may not construct a fully non-interactive EtF (i.e., without auxiliary committees); therefore, we look for efficient ways to construct EtF under standard assumptions while minimizing interaction. As a first step towards such a construction, we define the notion of Encryption to a Current Winner (ECW), which is a restricted version of EtF where messages can only be encrypted towards parties selected for a role whose lottery parameters are available for the *current* slot, the one in which we encrypt (this is as in previous constructions [BGG+20, GHK+21, GHM+21]). Unrestricted EtF, on the other hand, allows for encrypting a message toward lottery winners that will be determined at any arbitrary point in the future, including parties who only join the protocol execution far in the future (after the ciphertext has been generated).

Constructing ECW (non-interactively)—Section 6.4. We show that it is possible to construct a fully non-interactive ECW scheme from standard assumptions. Our construction relies on a milder flavor of witness encryption, which we call Witness Encryption over Commitments (cWE) and define it in Section 6.3. This primitive is significantly more restricted than full-fledged WE (see also discussion in Remark 6.2), but still powerful enough: we show in Section 6.4.1 that ECW can be constructed in a black-box manner from cWE, which in turn can be constructed from oblivious transfer and garbled circuits (Section 6.9). This construction improves over the previous results [BGG+20, GHK+21, GHM+21] since it does not rely on auxiliary committees.

Instantiating YOSO MPC using ECW—Section 6.5. The notion of ECW is more restricted than EtF, but it can still be useful in applications. We show how to use it as a building block for the YOSO MPC protocol of [GHK+21]. Here, each of the rounds in an MPC protocol is executed by a different committee. This same committee will simultaneously transfer its secret state to the next (near-future) committee, which in turn remains anonymous until it transfers its own secret state to the next committee, and so on. This setting clearly matches what ECW offers as a primitive, but it also introduces a few more requirements: 1. ECW ciphertexts must be non-malleable, *i.e.* we need an IND-CCA secure ECW scheme;

2. Only one party is selected for each role; 3. A party is selected for a role at random with probability proportional to its relative stake on the underlying PoS blockchain; 4. Parties selected for roles remain anonymous until they choose to reveal themselves; 5. A party selected for a role must be able to authenticate messages on behalf of the role, *i.e.* publicly proving that it was selected for a certain role and that it is the author of a message. We show that all of these properties can be obtained departing from an IND-CPA secure ECW scheme instantiated over a natural PoS blockchain (*e.g.* [DGKR18]). First, we observe that VRF-based lottery schemes implemented in many PoS blockchains are sufficient to achieve properties 2, 3 and 4. We then observe that natural block authentication mechanisms used in such PoS blockchains can be used to obtain property 5. Finally, we show that standard techniques can be used to obtain an IND-CCA secure ECW scheme from an IND-CPA secure ECW scheme.

Constructing EtF from ECW (interactively)—Section 6.6. Since we argued the implausibility of constructing EtF non-interactively from standard assumptions, we study how to transform an ECW scheme into an unrestricted EtF scheme when given access to an auxiliary committee but with "low communication" (and still from standard assumptions). We explain what we mean by "low communication" by an example of its opposite: in previous works ([BGG+20, GHK+21, GHM+21]) successive committees were required to store and reshare secret shares of every message to be sent to a party selected in the future. That is, their communication complexity grows both with the number and the amount and length of the encrypted messages. In contrast, our solution has communication complexity independent of the plaintext length. How our transformation from ECW to EtF works: we associate each role in the future to a unique identity of an Identity Based Encryption scheme (IBE); to encrypt a message towards a role we apply the encryption of the IBE scheme. When, at any point in the future, a party for that role is selected, a committee generates and delivers the corresponding secret key for that role/identity. To realize the latter step, we apply YOSO MPC instantiated from ECW as shown in Section 6.5. In contrast to previous schemes, our auxiliary committee only needs to hold shares of the IBE's master secret key and so it performs communication/computation dependent on the security parameter but not on the length/amount of messages encrypted to the future.

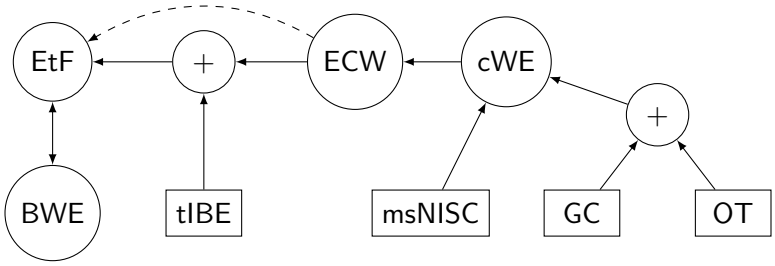


Figure 6.1: Dependency diagram for primitives in this work. Legend: primitives wrapped in circles are introduced in this work; $A \to B$: *"We can construct B from A"*; $A \dashrightarrow B$: *"A is a special case of B"*.

6.2 Modelling EtF

In this section, we present a model for encryption to the future winner of a lottery. In order to argue about a notion of future, we use the blocks of an underlying blockchain ledger and their relative positions in the chain to specify points in time. Intuitively, our notion allows for creating ciphertexts that can only be decrypted by a party that is selected to perform a certain role R at a future slot sl according to a lottery scheme associated with a blockchain protocol. The winner of the lottery at a point in the future with respect to a blockchain state $\tilde{\mathbf{B}}$ is determined by the lottery predicate defined in Section 4.2, *i.e.* the winner is the holder of a lottery secret key sk such that $\mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 1$. However, notice that the winner might only be determined by a blockchain state produced in the future as a result of the blockchain protocol execution. This makes it necessary for the ciphertext to encode an initial state $\mathbf{B}$ of the blockchain that allows for verifying that a future state $\tilde{\mathbf{B}}$ (presented at the time of decryption) has indeed been produced as a result of correct protocol execution. This requirement is captured by the evolving blockchain predicate defined in Section 4.2, *i.e.* $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ iff $\tilde{\mathbf{B}}$ is obtained as a future state of executing the blockchain protocol departing from $\mathbf{B}$.

Definition 6.1 (Encryption to the Future). *A pair of PPT algorithms* $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ *in the the context of a blockchain* Γ^V *is an EtF-scheme with evolved predicate* evolved *and a lottery predicate* lottery. *The algorithms work as follows.*

Encryption. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m)$ *takes as input an initial blockchain* $\mathbf{B}$*, a slot* sl*, a role* R *and a message* m*. It outputs a ciphertext* ct *- an encryption to the future.*

Decryption. $m/\perp \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk})$ *takes as input a blockchain state* $\tilde{\mathbf{B}}$*, a ciphertext* ct *and a secret key* sk *and outputs the original message* m *or* $\perp$.

An EtF must satisfy the following properties:

Correctness. An EtF-scheme is said to be correct if for honest parties i and j, there exists a negligible function μ such that for all sk, sl, R, m:

$$\left| \Pr \left[\begin{array}{c} \mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m) \\ \mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1 \end{array} : \begin{array}{c} \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 0 \\ \vee\ \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathsf{sk}) = m \end{array} \right] - 1 \right| \leq \mu(\lambda)$$

Security. We establish a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In Section 4.2 we describe how $\mathcal{A}$ and $\mathcal{Z}$ execute a blockchain protocol. In addition, we now let the adversary interact with the challenger in a game $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Algorithm 8. The game can be summarized as follows:

1. $\mathcal{A}$ executes the blockchain protocol Γ together with $\mathcal{Z}$ and at some round r chooses a blockchain $\mathbf{B}$, a role R for the slot sl and two messages m_0 and m_1 and sends it all to $\mathcal{C}$.

2. $\mathcal{C}$ chooses a random bit b and encrypts the message m_b with the parameters it received and sends ct to $\mathcal{A}$.

3. $\mathcal{A}$ continues to execute the blockchain until some round $\tilde{r}$ where the blockchain $\tilde{\mathbf{B}}$ is obtained and $\mathcal{A}$ outputs a bit b'.

If the adversary is a lottery winner for the challenge role R in slot sl, the game outputs a random bit. If the adversary is not a lottery winner for the challenge role R in slot sl, the game outputs $b \oplus b'$. The reason for outputting a random guess in the game when the challenge role is corrupted is as follows. Normally the output of the IND-CPA game is $b \oplus b'$ and we require it to be 1 with probability 1/2. This models that the guess b' is independent of b. This, of course, cannot be the case when the challenge role is corrupted. We therefore output a random guess in these cases. After this, any bias of the output away from 1/2 still comes from b' being dependent on b.

Algorithm 8: $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_{r}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ ▷ $\mathcal{A}$ executes Γ with $\mathcal{Z}$ until round r
$(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}})$ ▷ $\mathcal{A}$ outputs challenge parameters
$b \leftarrow\$ \{0,1\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_b)$
$\mathsf{st} \leftarrow \mathcal{A}(\mathsf{view}^r_{\mathcal{A}}, \mathsf{ct})$ ▷ $\mathcal{A}$ receives challenge ct
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$ ▷ Execute from view^r until round $\tilde{r}$
$(\tilde{\mathbf{B}}, b') \leftarrow \mathcal{A}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}}, \mathsf{st})$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ **then** ▷ $\tilde{\mathbf{B}}$ is a valid evolution of $\mathbf{B}$
  if $sk^{\mathcal{A}}_{L,j} \notin \mathcal{W}_{\tilde{\mathbf{B}},\mathsf{sl},\mathsf{R}}$ **then** ▷ $\mathcal{A}$ does not win role R
    return $b \oplus b'$
  end if
end if
return $\hat{b} \leftarrow\$ \{0,1\}$

Definition 6.2 (IND-CPA Secure EtF). *An EtF-scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ in the context of a blockchain protocol Γ executed by PPT machines $\mathcal{A}$ and $\mathcal{Z}$ is said to be* IND-CPA *secure if, for any $\mathcal{A}$ and $\mathcal{Z}$, there exists a negligible function μ such that for $\lambda \in \mathbb{N}$:*

$$\left| 2 \cdot \Pr\left[\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}} = 1 \right] - 1 \right| \leq \mu(\lambda)$$

Remark 6.1 (On the requirement of Proof-of-Stake for EtF). *The* EtF *notion requires the guarantee that an honest chain should be verifiable without interaction with the network (i.e. verified by the EtF ciphertext). While this is possible for Proof-of-Stake (PoS) blockchains, in a Proof-of-Work (PoW) blockchain the adversary can always simulate a chain where it generates all blocks. In general we require a blockchain in order to model time (via block height) for EtF.*

6.2.1 ECW as a Special Case of EtF

In this section we focus on a special class of EtF. We call schemes in this class ECW schemes. ECW is particularly interesting since the underlying lottery is always conducted with respect to the current blockchain state. This has the following consequences

1. $\mathbf{B} = \tilde{\mathbf{B}}$ means that $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ is trivially true.

2. The winner of role R in slot sl is already defined in $\mathbf{B}$.

It is easy to see that this kind of EtF scheme is simpler to realize since there is no need for checking if the blockchain has "correctly" evolved. Furthermore, all lottery parameters like stake distribution and randomness extracted from the blockchain are static. Thus, an adversary has no way to move stake between accounts in order to increase its chance of winning the lottery.
Note that, when using an ECW scheme, the lottery winner is already decided at encryption time. In other words, there is no delay and the moment a ciphertext is produced the receiver is chosen.

6.3 Witness Encryption over Commitments (cWE)

Here, we describe witness encryption over commitments that is a relaxed notion of witness encryption. In witness encryption parties encrypt to a public input for some NP statement. In cWE we have two phases: first parties provide a (honestly generated) commitment cm of their private input s. Later, anybody can encrypt to a public input for an NP statement which *also* guarantees correct opening of the commitment. Importantly, in applications, the first message in our model can be reused for many different invocations.

Remark 6.2 (Comparing cWE and WE)**.** *We observe that cWE is weaker than standard WE because of its deterministic flavor. In standard WE we encrypt without having any "pointer" to an alleged witness, but in cWE it requires the witness to be implicitly known at encryption time through the commitment (to which it is bound). That is why—as for the weak flavors of witness encryption in [BL20]—we believe it would be misleading to just talk about WE. This is true in particular since we show cWE can be constructed from standard assumptions such as oblivious transfer and garbled circuits (Section 6.9.2), whereas constructions of WE from standard assumptions are still an open problem or require strong primitives like indistinguishability obfuscation. Finally we stress a difference with the trivial "interactive" WE proposed in [GGSW13] (Section 1.3): cWE is still non-interactive after producing a once-and-for-all reusable commitment.*

6.3.1 Definition

The type of relations we consider are of the following form: a statement $\mathtt{x} = (\mathsf{cm}, C, y)$ and a witness $\mathtt{w} = (\mathsf{s}, \rho)$ are in the relation (i.e., $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$) iff "cm commits to some secret value s using randomness ρ, and $C(\mathsf{s}) = y$". Here, C is a circuit in some circuit class $\mathcal{C}$ and y is the expected output of the function.

Formally, we define witness encryption over commitments as follows:

Definition 6.3 (Witness encryption over commitments). *Let* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ *be a non-interactive commitment scheme. A cWE-scheme for witness encryption over commitments with circuit class* $\mathcal{C}$ *and commitment scheme* C *consists of a pair of algorithms* $\Pi_{\mathsf{cWE}} = (\mathsf{Enc}, \mathsf{Dec})$*:*

Encryption phase. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, \mathtt{x}, m)$ *on input a commitment key* ck*, a statement* $\mathtt{x} = (\mathsf{cm}, C, y)$ *such that* $C \in \mathcal{C}$*, and a message* $m \in \{0,1\}^*$*, generates a ciphertext* ct*.*

Decryption phase. $m/\bot \leftarrow \mathsf{Dec}(\mathsf{ck}, \mathsf{ct}, \mathtt{w})$ *on input a commitment key* ck*, a ciphertext* ct*, and a witness* w*, returns a message* m *or* $\bot$*.*

A cWE should satisfy *correctness* and *semantic security* as defined below.

(Perfect) Correctness. An honest prover with a statement $\mathtt{x} = (\mathsf{cm}, C, y)$ and witness $\mathtt{w} = (\mathsf{s}, \rho)$ such that $\mathsf{cm} = \mathsf{Commit}(\mathsf{ck}, \mathsf{s}; \rho)$ and $C(\mathsf{s}) = y$ can always decrypt with overwhelming probability. More precisely, a cWE with circuit class $\mathcal{C}$ and commitment scheme C has perfect correctness if for all $\lambda \in \mathbb{N}$, $C \in \mathcal{C}$, $\mathsf{ck} \in \mathsf{Range}(\mathsf{C.Setup})$, $\mathsf{s} \in \mathcal{S}_m$, randomness $\rho \in \mathcal{S}_r$, commitment $\mathsf{cm} \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho)$, and bit message $m \in \{0,1\}^*$, it holds that

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, C(\mathsf{s})), m); m' \leftarrow \mathsf{Dec}(\mathsf{ck}, \mathsf{ct}, (\mathsf{s}, \rho)) \;:\; m = m'\right] = 1$$

(Weak) Semantic Security. Intuitively, encrypting with respect to a false statement (with honest commitment) produces indistinguishable ciphertexts. Formally, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all auxiliary strings aux and all PPT adversaries $\mathcal{A}$:

$$\left| 2 \cdot \Pr\left[\begin{array}{r} \mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ (\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1) \leftarrow \mathcal{A}(\mathsf{ck}, \mathsf{aux}) \\ \mathsf{cm} \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho); b \leftarrow\$ \{0,1\} \;:\; \mathcal{A}(\mathsf{st}, \mathsf{ct}) = b \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, y), m_b) \\ \mathsf{ct} := \bot \text{ if } C(\mathsf{s}) = y,\ C \notin \mathcal{C} \text{ or } |m_0| \neq |m_1| \end{array}\right] - 1 \right| \leq \mu(\lambda)$$

Later, to show the construction of ECW from cWE, we need a stronger notion of semantic security where the adversary additionally gets to see ciphertexts of the challenge message under true statements with unknown to $\mathcal{A}$ witnesses. We formalize this property in Section 6.11.1 and show that weak semantic security together with hiding of the commitment imply strong semantic security.

6.3.2 Constructions of cWE

From Multi-Sender 2P-NISC [AMPR14]. A cWE scheme can be constructed from protocols for Multi-Sender (reusable) Non-Interactive Secure Computation (MS-NISC) [AMPR14]. In such protocols, there is a receiver R with input x who first broadcasts an encoding of its input, and then later every sender S_i with input y_i can send a single message to R that conveys only $f(x, y_i)$. This is done while preserving privacy of inputs and correctness of output.

In Section 6.9.1 we provide a detailed explanation of how to construct cWE using MS-NISC as in [AMPR14]. We here state the main points of the construction.
Let f be the function that on input $y = (\mathtt{x}, k)$ and $x = \mathtt{w}$ outputs k if and only if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$. This will be the underlying function for the MS-NISC protocol. We then obtain a cWE scheme over the relation $\mathcal{R}$ in the following way:

1. First, the receiver commits to its witness $\mathtt{w}$ by providing an encoding of it as its first message in the MS-NISC protocol.

2. Secondly, to encrypt m under statement $\mathtt{x}$, a sender samples a key k of size $|m|$ and provides an encoding of $(\mathtt{x}, k)$ as the second message in the MS-NISC protocol and sends the ciphertext $\mathsf{ct} = m \oplus k$ to the receiver.

3. Finally, the receiver obtains the key as the output of $f(x = \mathtt{w}, y = (\mathtt{x}, k)) = k$ iff $\mathtt{w}$ is a valid witness for the statement $\mathtt{x}$ encoded in the second message. And it decrypts the ciphertext $m = \mathsf{ct} \oplus k$.

We observe that the above construction actually yields a stronger notion of cWE where the statement $\mathtt{x}$ is private which is not a requirement in our setting. This asymmetry between sender and receiver privacy was also observed by others [JKO13] and it opens the door for efficient constructions using oblivious transfer (OT) and privacy-free garbled circuits as described in [ZRE15]. More details on the more efficient construction of cWE using OT and garbled circuits are provided in Section 6.9.2.

6.4 Construction of ECW

Here we show a novel construction of ECW from cWE. We then show alternative constructions through instantiations from previous work.

6.4.1 ECW from cWE

In this section we realize the notion of ECW from cWE. We define our scheme with respect to a set of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ executing a blockchain protocol Γ as described in Section 4.2, *i.e.* each party P_i has access to the blockchain ledger and is associated

to a tuple $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{st}_i)$ registered in the genesis block for which it has corresponding secret keys $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$. Our construction uses as a main building block a witness encryption scheme over commitments $\Pi_{\mathsf{cWE}} = (\mathsf{Enc}_{\mathsf{cWE}}, \mathsf{Dec}_{\mathsf{cWE}})$; we assume the commitments to be extractable. The class of circuits $\mathcal{C}$ of Π_{cWE} includes the lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$. We let each party publish an initial commitment of its witness. This way we can do without any interaction for encryption/decryption through a one-time setup where parties publish the commitments over which all following encryptions are done. We construct our ECW scheme Π_{ECW} as follows:

System Parameters: We assume that a commitment key $\mathsf{Setup}(1^\lambda) \to \mathsf{ck}$ is contained in the genesis block B_0 of the underlying blockchain.

Setup Phase: All parties $P_i \in \mathcal{P}$ proceed as follows:

1. Compute a commitment $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$ to $\mathsf{sk}_{L,i}$ using randomness ρ_i. We abuse the notation and define P_i's secret key as $\mathsf{sk}_{L,i}||\rho_i$.

2. Compute a signature $\sigma_i \leftarrow \mathsf{Sig}_{\mathsf{Sig.sk}_i}(\mathsf{cm}_i)$.

3. Publish $(\mathsf{cm}_i, \sigma_i)$ on the blockchain by executing $\mathsf{Broadcast}(1^\lambda, (\mathsf{cm}_i, \sigma_i))$.

Encryption $\mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m)$**:** Construct a circuit C that encodes the predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$, where $\mathbf{B}$, sl and R are hardcoded and $\mathsf{sk}_{L,i}$ is the witness. Let $\mathcal{P}_{Setup}$ be the set of parties with non-zero relative stake and a valid setup message $(\mathsf{cm}_i, \sigma_i)$ published in the common prefix $\mathbf{B}^{\lceil\kappa}$ (if P_i has published more than one valid $(\mathsf{cm}_i, \sigma_i)$, only the latest one is considered). For every $P_i \in \mathcal{P}_{Setup}$, compute $\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{cWE}}(\mathsf{ck}, \mathsf{x}_i = (\mathsf{cm}_i, C, 1), m)$. Output $\mathsf{ct} = \left(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \{\mathsf{ct}_i\}_{P_i \in \mathcal{P}_{Setup}}\right)$.

Decryption $\mathsf{Dec}(\mathbf{B}, \mathsf{ct}, \mathsf{sk})$**:** Given $\mathsf{sk} := \mathsf{sk}_{L,i}||\rho_i$ such that $\mathsf{cm}_i = \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,i}; \rho_i)$ and $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$ for parameters $\mathbf{B}, \mathsf{sl}, \mathsf{R}$ from ct, output $m \leftarrow \mathsf{Dec}_{\mathsf{cWE}}(\mathsf{ck}, \mathsf{ct}_i, (\mathsf{sk}_{L,i}, \rho_i))$. Otherwise, output $\perp$.

Theorem 6.1. *Let* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ *be a non-interactive extractable commitment scheme and* $\Pi_{\mathsf{cWE}} = (\mathsf{Enc}_{\mathsf{cWE}}, \mathsf{Dec}_{\mathsf{cWE}})$ *be a strong semantically secure cWE over* C *for a circuit class* $\mathcal{C}$ *encoding the lottery predicate* $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ *as defined in Section 6.3. Let* Γ *be a blockchain protocol as defined in Section 4.2.* Π_{ECW} *is an* IND-CPA*-secure ECW scheme as per Definition 6.2.*

The proof is provided in Section 6.11.2.

6.4.2 Other Instantiations

ECW from target anonymous channels [GHM+21, BGG+20]. As mentioned before, another approach to construct ECW can be based on a recent line of work that aims to design secure-MPC protocols where parties should remain anonymous until they

speak [GHM+21, BGG+20, GHK+21]. The baseline of these results is to establish a communication channel to random parties, while preserving their anonymity. It is quite clear that such anonymous channels can be used to realize our definition of ECW for the underlying lottery predicate that defines to whom the anonymous channel is established. Namely, to encrypt m to a role R at a slot sl with respect to a blockchain state $\mathbf{B}$, create a target anonymous channel to $(\mathsf{R}, \mathsf{sl})$ over $\mathbf{B}$ by using the above approaches and send m via this channel. Depending on the lottery predicate that specifies which random party the channel is created for, a recipient with the secret key who wins this lottery can retrieve m. To include some concrete examples, the work of Benhamouda et al. [BGG+20] proposed the idea of using a "nomination" process, where a nominating committee chooses a number of random parties $\mathcal{P}$, look up their public keys, and publish a re-randomization of their key. This allows everyone to send messages to $\mathcal{P}$ while keeping their anonymity. The work of [GHM+21] answered this question differently by delegating the nomination task to the previous committees without requiring a nominating committee. That is, the previous committee runs a secure-MPC protocol to choose a random subset of public keys, and broadcasts the rerandomization of the keys. To have a MPC protocol that scales well with the total number of parties, they define a new flavour of private information retrieval (PIR) called random-index PIR (or RPIR) and show how each committee—playing the role of the RPIR client—can select the next committee with the complexity only proportional to the size of the committee. There are two constructions of RPIR proposed in [GHM+21], one based on Mix-Nets and the other based on FHE. Since the purpose of the constructions described is to establish a target-anonymous channel to a random party, one can consider them as examples of a stronger notion of ECW with anonymity and a specific lottery predicate that selects *a single* random party from the entire population as the winner.

ECW from [DS15]. Derler and Slamanig [DS15] (DS) constructed a variant of WE for a restricted class of algebraic languages. In particular, a user can conduct a Groth-Sahai (GS) proof for the satisfiability of some pairing-product equations (PPEs). Such a proof contains commitments to the witness using randomness only known by this user. The proof can be used by anyone to encrypt a message resulting in a ciphertext which can only be decrypted by knowing this randomness. More formally, they consider a type of WE associated with a proof system $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ consisting of two rounds. In the first round, a recipient computes and broadcasts $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w})$. Later, a user can verify the proof and encrypt a message m under $(\mathtt{x}, \pi)$ if $\mathsf{Verify}(\mathsf{crs}, \mathtt{x}, \pi) = 1$. We note that the proof π does not betray the user conducting the proof and therefore it can use an anonymous broadcast channel to communicate the proof to the encrypting party in order to obtain anonymous ECW. Moreover, although GS proofs may look to support only a restricted class of statements based on PPEs, they are expressive enough to cover all the statements arising in pairing-based cryptography. This indicates the applicability of this construction for any VRF-based lottery where the VRF is algebraic and encodable as a set of PPEs. Further details are provided in Section 6.8. This interactive ECW just described yields an improvement in communication complexity at the cost of having an extra round of interaction.

From Signatures of Knowledge. Besides the above instantiations, we point out a (potentially more inefficient) abstract construction from zero-knowledge signatures of knowledge (SoK) [CL06] (roughly, a non-malleable non-interactive zero-knowledge proof). This is similar in spirit to the previous instantiation and can be seen as a generalization. As-

sume each party has a (potentially ephemeral) public key. At the time the lottery winner has been decided, the winners can post a SoK showing knowledge of the secret key corresponding to their pk *and* that their key is a winner of the lottery. To encrypt, one would first verify the SoK and then encrypt with respect to the corresponding public key.

6.5 YOSO Multiparty Computation from ECW

In this section we show how ECW can be used as the crucial ingredient in setting up a YOSO MPC. So far we have only focused on IND-CPA secure ECW, which falls short of role assignment in the sense of [GHK+21]. In general role assignment requires the following properties which are not provided by ECW (or EtF):

1. Multiple parties must be able to send messages to the same role (in most applications this requires IND-CCA).

2. Parties must authenticate messages on behalf of a role they executed in the past (authentication from the past)

3. A party assigned to a given role must stay covert until the role is executed.

We will define a number of properties needed for EtF to realize applications such as role assignment. We start by looking at CCA security for an EtF scheme. We then introduce the notion of Authentication from the Past (AfP) and definition of unforgeability and privacy guarantees. Finally, we introduce the notion of YOSO-friendly blockchains that have in-built lotteries with properties that are needed to conduct YOSO MPC and corresponding EtF and AfP schemes.

6.5.1 IND-CCA EtF

In this section we define what it means for an EtF to be IND-CCA secure. This security property is useful in many applications where more encryptions are done towards the same slot and role. As in the definition of IND-CPA, we establish a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We introduce a decryption oracle, $\mathcal{O}_{\mathsf{EtF}}$, which on input ct returns the decryption of ciphertext. Furthermore, the $\mathcal{O}_{\mathsf{EtF}}$ maintains a list of ciphertext queries $\mathcal{Q}_{\mathsf{EtF}}$. Algorithm 9 shows the details of the game.

Definition 6.4 (IND-CCA2 Secure EtF)**.** *Formally, an EtF-scheme $\mathcal{E}$ is said to be IND-CCA2 secure in the context of a blockchain protocol Γ executed by PPT machines $\mathcal{A}$ and $\mathcal{Z}$ if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$:*

$$\left|2 \cdot \Pr\left[\mathsf{Game}^{\text{IND-CCA2}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}} = 1\right] - 1\right| \leq \mu(\lambda)$$

To add IND-CCA2 security to an IND-CPA secure EtF scheme (as defined in Definition 6.2) we can use standard transformations such as [FO99, Sah99]. In the transformation based

Algorithm 9: $\mathsf{Game}^{\text{IND-CCA2}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathsf{E}}$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_{r}(\mathcal{A}^{\mathcal{O}_{\mathsf{EtF}}}, \mathcal{Z}, 1^{\lambda})$ ▷ $\mathcal{A}$ executes Γ with $\mathcal{Z}$ until round r
$(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{EtF}}}(\mathsf{view}^r_{\mathcal{A}})$ ▷ $\mathcal{A}$ outputs challenge parameters
$b \leftarrow\$ \{0,1\}$
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_b)$
$\mathsf{st} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{EtF}}}(\mathsf{view}^r_{\mathcal{A}}, \mathsf{ct})$ ▷ $\mathcal{A}$ receives challenge ct
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}^{\mathcal{O}_{\mathsf{EtF}}}, \mathcal{Z}, 1^{\lambda})$ ▷ Execute from view^r until round $\tilde{r}$
$(\tilde{\mathbf{B}}, b') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{EtF}}}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}}, \mathsf{st})$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ **then** ▷ $\tilde{\mathbf{B}}$ is a valid evolution of $\mathbf{B}$
  if $\mathsf{sk}^{\mathcal{A}}_{L,j} \notin \mathcal{W}_{\tilde{\mathbf{B}},\mathsf{R},\mathsf{sl}} \wedge \mathsf{ct} \notin \mathcal{Q}_{\mathsf{EtF}}$ **then** ▷ $\mathcal{A}$ does not win role R
    return $b \oplus b'$
  end if
end if
return $g \leftarrow\$ \{0,1\}$

on [Sah99] we could add to the setup of the blockchain a CRS for a simulation-sound extractable NIZK. When encrypting m to a role R the sender will send along a proof of knowledge of the plaintext m. We get the challenge ciphertext from the IND-CPA game and use the ZK property to simulate the NIZK proof. We can use the extraction trapdoor of the proof system to simulate the CCA decryption oracles by simulation soundness. When the IND-CCA2 adversary makes a guess, we make the same guess. The details of the construction and proof follow using standard techniques and are omitted. On the other hand, the popular transformation of [FO99] allows for simulating CCA decryption oracles by observing the adversary's queries to a random oracle, which should not be an issue since an EtF scheme is likely already running on top of a blockchain which is secure in the random oracle model. We leave the construction of concretely efficient IND-CCA2 EtF as future work.

6.5.2 Authentication from the Past (AfP)

When the winner of a role R_1 sends a message m to a future role R_2 then it is typically also needed that R_2 can be sure that the message m came from a party P which, indeed, won the role R_1. Most PoS blockchains deployed in practice have a lottery where a certificate can be released proving that P won the role R_1. In order to formalize this concept, we introduce an AfP scheme with a corresponding EUF-CMA game representing the authentication property.

Definition 6.5 (Authentication from the Past)**.** *A pair of PPT algorithms* $\mathcal{U} = (\mathsf{Sign}, \mathsf{Verify})$ *is a scheme for authenticating messages as a winner of a lottery in the past in the context of blockchain* Γ *with lottery predicate* $\mathsf{lottery}$*.*

Authenticate. $\sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}, m)$ *takes as input a blockchain* $\mathbf{B}$*, a slot* sl*, a role* R*, a secret key* sk*, and a message* m*. It outputs a signature* σ *that authenticates the message* m*.*

Verify. $\{0,1\} \leftarrow \mathsf{AfP.Verify}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}, \sigma, m)$ *uses the blockchain* $\tilde{\mathbf{B}}$ *to ensure that* σ *is a signature on* m *produced by the secret key winning the lottery for slot* sl *and role* R.

Furthermore, an AfP-scheme has the following properties:

Correctness. *An AfP-scheme is said to be correct if for honest parties* i *and* j*, there exists a negligible function* μ *such that for all* sk, sl, R, m*:*

$$\left| \Pr \left[\begin{array}{c} \mathsf{view} \leftarrow \mathit{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda}) \\ \mathbf{B} = \mathsf{GetRecords}(\mathsf{view}_i) \\ \tilde{\mathbf{B}} = \mathsf{GetRecords}(\mathsf{view}_j) \\ \sigma \leftarrow \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}, m) \end{array} : \begin{array}{c} \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 0 \\ \vee\ \mathsf{lottery}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 0 \\ \vee\ \mathsf{AfP.Verify}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}, \sigma, m) = 1 \end{array} \right] - 1 \right| \leq \mu(\lambda)$$

In other words, an AfP on a message from an honest party with a view of the blockchain $\mathbf{B}$ *can attest to the fact that the sender won the role* R *in slot* sl*. If another party, with blockchain* $\tilde{\mathbf{B}}$ *agrees, then the verification algorithm will output 1.*

Security. *We here describe the game detailed in Algorithm 10 representing the security of an AfP scheme. The algorithm represents a standard EUF-CMA game where the adversary has access to a signing oracle* $\mathcal{O}_{\mathsf{AfP}}$ *which it can query with a slot* sl*, a role* R *and a message* m_i *and obtain AfP signatures* $\sigma_i = \mathsf{AfP.Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_j, m_i)$ *where* $\mathsf{sk}_j \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{R}}$ *i.e.* $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_j) = 1$*. The oracle maintains the list of queries* $\mathcal{Q}_{\mathsf{AfP}}$*.*
Formally, an AfP-scheme $\mathcal{U}$ *is said to be EUF-CMA secure in the context of a blockchain protocol* Γ *executed by PPT machines* $\mathcal{A}$ *and* $\mathcal{Z}$ *if there exists a negligible function* μ *such that for* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}} = 1\right] \leq \mu(\lambda)$$

Algorithm 10: $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$

$\mathsf{view} \leftarrow \mathsf{EXEC}^{\Gamma}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ $\mathcal{A}$ executes Γ with $\mathcal{Z}$
$(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m', \sigma') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}_{\mathcal{A}})$
if $(m' \in \mathcal{Q}_{\mathsf{AfP}}) \vee (\mathsf{sk}^{\mathcal{A}}_{L,j} \in \mathcal{W}_{\mathbf{B},\mathsf{sl},\mathsf{R}})$ **then** ▷ $\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}$ won or queried illegal m'
  return 0
end if
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^{r},\tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^{\lambda})$ ▷ Execute from view^{r} until round $\tilde{r}$
$\tilde{\mathbf{B}} \leftarrow \mathsf{GetRecords}(\mathsf{view}^{\tilde{r}}_i)$
if $\mathsf{evolved}(\mathbf{B}, \tilde{\mathbf{B}}) = 1$ **then**
  if $\mathsf{AfP.Verify}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \sigma', m') = 1$ **then** ▷ $\mathcal{A}$ successfully forged an AfP
    return 1
  end if
end if
return 0

6.5.2.1 General AfP.

In general we can add authentication to a message as follows. Recall that P_i wins R if $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$. Here, $\mathcal{R}(\mathtt{x} = (\mathbf{B}, \mathsf{sl}, \mathsf{R}), \mathtt{w}) = \mathsf{lottery}(\mathtt{x}, \mathtt{w})$ is an NP relation where all parties know $\mathtt{x}$ but only the winner knows a witness $\mathtt{w}$ such that $\mathcal{R}(\mathtt{x}, \mathtt{w}) = 1$. We can therefore use a signature of knowledge (SoK) [CL06] to sign m under the knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$. This will attest that the message m was sent by a winner of the lottery for R. In Section 6.12.1, we show more efficient construction of AfP by exploring the structure of PoS-based blockchains with VRF lotteries.

6.5.3 AfP Privacy

Just EUF-CMA security is not sufficient for an AfP mechanism to be YOSO friendly. It must also preserve the privacy guarantees of the lottery predicate, guaranteeing that the adversary does not gain any undue advantage in predicting when a party is selected to perform a role after it uses AfP to authenticate a message. To appreciate this fact, we consider the case where instead of creating a signature of knowledge of $\mathsf{sk}_{L,i}$ on message m we simply use a regular EUF-CMA secure signature scheme to sign the message concatenated with $\mathsf{sk}_{L,i}$, revealing the signature public key, the resulting signature and $\mathsf{sk}_{L,i}$ itself as a means of authentication. By definition, this will still constitute an existentially unforgeable AfP but will also reveal whether the party who owns $\mathsf{sk}_{L,i}$ is the winner when future lotteries are conducted. The specific privacy property we seek is that an adversary, observing AfP tags from honest parties, cannot use this information to enhance its chances in predicting the winners of lotteries for roles for which an AfP tag has not been published. On the other hand, the identity of a party who won the lottery for a given role is not kept private when it publishes an AfP tag on behalf of this role, which is not an issue in a YOSO-setting since corruption after-the-fact is futile. Specifically, we allow an AfP tag to be linked to the identity of the party who generated it. Note, that this kind of privacy is different from notions like k-anonymity since the success of the adversary in guessing lottery winners with high accuracy depends on the stake distribution. The stake distribution is public in most PoS-settings and, thus, a privacy definition must take into account this inherent leakage.

Definition 6.6 (AfP Privacy.)**.** *An AfP scheme $\mathcal{U}$ with corresponding lottery predicate* lottery *is private if a PPT adversary $\mathcal{A}$ is unable to distinguish between the scenarios defined in Algorithm 11 and Algorithm 12 with more than negligible probability in the security parameter.*

Scenario 0 ($b = 0$). *In this scenario (Algorithm 11), $\mathcal{A}$ is first running the blockchain Γ together with the environment $\mathcal{Z}$. At round r, $\mathcal{A}$ is allowed to interact with the oracle $\mathcal{O}_{\mathsf{AfP}}$ (see Definition 6.5). The adversary then continues the execution until round $\tilde{r}$ where it outputs a bit b'.*

Scenario 1 ($b = 1$). *This scenario (Algorithm 12) is identical to scenario 0 but instead of interacting with $\mathcal{O}_{\mathsf{AfP}}$, the adversary interacts with a simulator* Sim.

Algorithm 11: $b = 0$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
$\mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}^r_{\mathcal{A}})$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AfP}}}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}})$

Algorithm 12: $b = 1$

$\mathsf{view}^r \leftarrow \mathsf{EXEC}^{\Gamma}_r(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
$\mathcal{A}^{\mathsf{Sim}}(\mathsf{view}^r_{\mathcal{A}})$
$\mathsf{view}^{\tilde{r}} \leftarrow \mathsf{EXEC}^{\Gamma}_{(\mathsf{view}^r, \tilde{r})}(\mathcal{A}, \mathcal{Z}, 1^\lambda)$
return $b' \leftarrow \mathcal{A}^{\mathsf{Sim}}(\mathsf{view}^{\tilde{r}}_{\mathcal{A}})$

We let $\mathsf{Game}^{\mathsf{AfP\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$ *denote the game where a coin-flip decides whether the adversary is executed in scenario 0 or scenario 1. We say that the adversary wins the game (i.e.* $\mathsf{Game}^{\mathsf{AfP\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}} = 1$*) iff* $b' = b$*. Finally, an AfP scheme* $\mathcal{U}$ *is called private in the context of the blockchain* Γ *executed together with environment* $\mathcal{Z}$ *if the following holds for a negligible function* μ*.*

$$\left|2 \cdot \Pr\left[\mathsf{Game}^{\mathsf{AfP\text{-}PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}} = 1\right] - 1\right| \leq \mu(\lambda)$$

6.5.4 Round and Committee Based YOSO Protocols

Having IND-CCA2 ECW and an EUF-CMA secure and Private AfP, we can establish a round-based YOSO model, where there is a number of rounds $r = 1, 2, \ldots$ and where for each round there are n roles $\mathsf{R}_{r,i}$. We call the role $\mathsf{R}_{r,i}$ "party i in round r". We fix a round length L and associate role $\mathsf{R}_{r,i}$ to slot $\mathsf{sl} = L \cdot r$. This L has to be long enough that in each round the parties executing the roles can decrypt ciphertexts sent to them, execute the steps of the role, compute encryptions to the roles in the next round and post these to the blockchain in time for these to be available to the committee of round $r+1$ before slot $(r+1) \cdot L$. Picking such an L depends crucially on the underlying blockchain and network, and we will here simply assume that it can be done for the blockchain at hand.

Using this setup, the roles $\mathsf{R}_{r,i}$ of round r can use ECW and AfP with the aforementioned properties to send secret authenticated messages to the roles $\mathsf{R}_{r+1,i}$ in round $r+1$. They find their ciphertexts on the blockchain before slot $r \cdot L$, decrypt using ECW, compute their outgoing messages, encrypt using ECW, authenticate using AfP, and post the ciphertexts and AfP tags on the blockchain.

6.5.4.1 Honest Majority.

In round based YOSO MPC it is critical that we can assume some fraction of honesty in each committee $\mathsf{R}_{r,1}, \ldots, \mathsf{R}_{r,n}$. We discuss here assumptions needed on the lottery for this to hold and how to guarantee it. Assume an adversary that can corrupt parties identified by sk and a lottery assigning parties to roles $\mathsf{R}_{r,i}$. We map the corruption status of parties to roles as follows:

1. If a role $\mathsf{R}_{r,i}$ is won by a corrupted party or by several parties, call the role MALICIOUS. Even if $\mathsf{R}_{r,i}$ is won by two honest parties, they will both execute the role and send outgoing messages, which might violate security.

2. If a role $\mathsf{R}_{r,i}$ is won by exactly one honest party, call it HONEST.

3. If a role $\mathsf{R}_{r,i}$ is not won by any party, call it CRASHED. These roles will not be executed and are therefore equivalent to a crashed party.

Note that because we assume corrupted parties know their lottery witness $\mathsf{sk}_{L,i}$ in our model, we can, in poly-time, extract those witnesses and compute the corruption status of roles. This will be crucial in our reductions. Imagine that a role could be won by an honest party but also by a corrupted party which stays completely silent but decrypts messages sent to the role. If we are not aware of the corrupted party winning the role, we might send a simulated ciphertext to the apparently honest role. The corrupted party also having won the role would be able to detect this. Since any role won by an honest party could also be corrupted by a silent malicious party, simulation would become impossible.

In order to realize YOSO MPC, we will need committees where a majority of the roles are honest according to the description above. We capture this requirement in the definition below and argue how it can be achieved in Section 6.12.2.

Definition 6.7 (Honest Committee Friendly). *We call a blockchain Γ honest committee friendly if there exist n and H and T such that $H > T$ s.t. we can define a sequence of roles $\mathsf{R}_{r,i}$ for $r = 1, \dots, \mathsf{poly}(\lambda)$ and $i = 1, \dots, n$ for a slot sl_r and that for all r it holds that except with negligible probability there are at least H honest roles in $\mathsf{R}_{r,1}, \dots, \mathsf{R}_{r,n}$ and at most T malicious roles. Furthermore, if an honest party executing $\mathsf{R}_{r,1}, \dots, \mathsf{R}_{r,n}$ sends a message at sl_r, it is guaranteed to appear on the blockchain before slot sl_{r+1}.*

We are now ready to capture the above discussion using a definition.

Definition 6.8 (YOSO Friendly Blockchain). *Let Γ be a blockchain with a lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_r, \mathsf{R}_{r,i}, \mathsf{sk}_{L,i})$ and let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ and $\mathcal{U} = (\mathsf{Sign}, \mathsf{Verify})$ be an EtF and AfP for $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_r, \mathsf{R}_{r,i}, \mathsf{sk}_{L,i})$, respectively. We call $(\Gamma, \mathcal{E}, \mathcal{U})$ YOSO MPC friendly if the following holds:*

1. *$\mathcal{E}$ is an IND-CCA2 secure EtF (Definition 6.4).*

2. *$\mathcal{U}$ is a secure and private AfP (Definition 6.5 and Definition 6.6).*

3. *Γ is honest committee friendly (Definition 6.7).*

We will later assume a YOSO friendly blockchain, and we argued above that the existence of a YOSO friendly blockchain is a plausible assumption without having given formal proofs of this. It is interesting future work to prove that a concrete blockchain is a YOSO friendly blockchain in a given communication model. We omit this as our focus is on constructing flavours of EtF.

6.6 Construction of EtF from ECW and Threshold-IBE

The key intuition about our construction is as follows: we use IBE to encrypt messages to an arbitrary future $(\mathsf{R}, \mathsf{sl}_{\mathsf{fut}})$ pair. When the winners of the role in slot $\mathsf{sl}_{\mathsf{fut}}$ are assigned, we let them obtain an ID-specific key for $(\mathsf{R}, \mathsf{sl}_{\mathsf{fut}})$ from the IBE key-generation algorithm using ECW as a channel. Notice that this key-generation happens in the *present* while the encryption could have happened at any earlier time. We generate the key for $(\mathsf{R}, \mathsf{sl}_{\mathsf{fut}})$ in a threshold manner by assuming that, throughout the blockchain execution, a set of committee members each holds a share of the master secret key msk_i.

6.6.1 Construction

We now describe our construction. We assume an encryption to the current winner $\Pi_{\mathsf{ECW}} = (\mathsf{Enc}_{\mathsf{ECW}}, \mathsf{Dec}_{\mathsf{ECW}})$ and a threshold IBE scheme Π_{TIBE}. In the setup stage we assume a dealer acting honestly by which we can assign master secret key shares of the TIBE.

Parameters: We assume that the genesis block B_0 of the underlying blockchain contains all the parameters for Π_{ECW}.

Setup Phase: Parties run the setup stage for the Π_{ECW}. The dealer produces $(\mathsf{mpk}, \vec{\mathsf{msk}} = (\mathsf{msk}_1, \ldots, \mathsf{msk}_n))$ from TIBE setup with threshold k. Then it chooses n random parties and gives a distinct msk_i to each. All learn mpk.

Blockchain Execution: The blockchain execution we assume is as in Section 6.2. We additionally require that party i holding a master secret key share msk_i broadcasts $\mathsf{ct}^{\mathsf{sk},i}_{(\mathsf{sl},\mathsf{R})} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}^i_{(\mathsf{sl},\mathsf{R})})$, whenever the winner of role R in slot sl is defined in the blockchain $\mathbf{B}$, where $\mathsf{sk}^i_{(\mathsf{sl},\mathsf{R})} \leftarrow \Pi_{\mathsf{TIBE}}.\mathsf{IDKeygen}(\mathsf{msk}_i, (\mathsf{sl}, \mathsf{R}))$.

Encryption $\mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m)$**:** Each party generates $\mathsf{ct}_i \leftarrow \Pi_{\mathsf{TIBE}}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{ID} = (\mathsf{sl}, \mathsf{R}), m)$. Output $\mathsf{ct} = (\mathbf{B}, \mathsf{sl}, \mathsf{R}, \{\mathsf{ct}_i\}_{P_i})$.

Decryption $\mathsf{Dec}(\mathbf{B}, \mathsf{ct}, \mathsf{sk})$**:** Party i outputs $\perp$ if it does not have $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$ for parameters $\mathbf{B}, \mathsf{sl}, \mathsf{R}$ from ct. Otherwise, it retrieves enough (above threshold) valid ciphertexts $\mathsf{ct}^{\mathsf{sk},j}_{(\mathsf{sl},\mathsf{R})}$ from the current state of the blockchain and decrypts each through Π_{ECW} obtaining $\mathsf{sk}^j_{(\mathsf{sl},\mathsf{R})}$. It then computes $\mathsf{sk}_{(\mathsf{sl},\mathsf{R})} \leftarrow \Pi_{\mathsf{TIBE}}.\mathsf{Combine}(\mathsf{mpk}, (\mathsf{sk}^j_{(\mathsf{sl},\mathsf{R})})_j)$. It finally outputs $m \leftarrow \Pi_{\mathsf{TIBE}}.\mathsf{Dec}(\mathsf{sk}_{(\mathsf{sl},\mathsf{R})}, \mathsf{ct})$.

Resharing. We can ensure that the master secret key is proactively reshared by modifying each party so that msk_i-s are reshared and reconstructed in the evolution of the blockchain.

Correctness. Correctness of the construction follows from the correctness of the underlying IBE and the fact that a winning role will be able to decrypt the id-specific key by the correctness of the ECW scheme.

6.6.2 Security and Proof Intuition

In the following we assume some of the extensions discussed in Section 6.5.

Theorem 6.2 (informal). *Let Γ^V be a YOSO MPC friendly blockchain, Π_{TIBE} be a robust secure threshold IBE as in Section 5.5.2 with threshold $n/2$, and Π_{ECW} a secure* IND-CCA2 *ECW. The construction in Section 6.6.1 is a secure EtF.*

We describe our proof in Section 6.10. At the high level we show security in two steps. We first show the security of our construction for a simplified non-threshold setting with a standard IBE instead of a threshold one with key-sharing. In other words we do not temporarily consider the real case where there is a committee of parties holding a share of the master secret key, but we assume the execution uses a "key provider" oracle holding the master secret key of the IBE scheme. In particular, we define the behavior of oracle $\mathcal{O}^{\text{k-provider}}_{\mathsf{msk}}$ as follows: given in input a blockchain $\mathbf{B}$ and a slot sl (such that the latest slot of $\mathbf{B}$ is sl), it broadcasts a ciphertext for the winner[2] of the slot computed as $\mathsf{ct}^{\mathsf{sk}}_{\mathsf{sl}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{\mathsf{sl}})$ where $\mathsf{sk}_{\mathsf{sl}} \leftarrow \mathsf{IBE.Keygen}(\mathsf{msk}, (\mathsf{sl}, \mathsf{R}))$.

As a second step in the proof we show that, in the threshold-setting (where the master secret key is actually shared), one can obtain an adversary with a comparable advantage in the threshold-setting from an adversary in the non-threshold setting. Intuitively, we can do this because of the low amount of stake the adversary is controlling and the security of threshold-IBE.

Finally, our proof considers the case of an adversary with static corruptions, but we point out it can be straightforwardly compiled to a full round and committee YOSO setting as described in Section 6.5.

6.7 Blockchain WE versus EtF

In this section we show that an account-based PoS blockchain with sufficiently expressive smart contracts and an EtF scheme for this blockchain implies a notion of witness encryption on blockchains, and *vice versa*. The construction of EtF from BWE is completely straightforward and natural: encrypt to the witness which is the secret key winning the lottery. The construction of BWE from EtF is also straightforward but slightly contrived: it requires that we can restrict the lottery such that only some accounts can win a given role and that the decryptor has access to a constant fraction of the stake on the blockchain

[2] This is actually a vector, one for each winner in the slot. For clarity of discussion we just consider the case for one winner. The general case follows straightforwardly.

and are willing to bind them for the decryption operation. The reason why we still prove the result is that it establishes a connection at the feasibility level. For sufficiently expressive blockchains the techniques allowing to construct EtF and BWE are the same. To get EtF from simpler techniques than those we need for BWE we need to do it in the context of very simple blockchains. In addition, the techniques allowing to get EtF without getting BWE should be such that they prevent the blockchain from having an expressive smart contract layer added. This seems like a very small loophole, so we believe that the result shows that there is essentially no assumptions or techniques which allow to construct EtF which do not also allow to construct BWE. Since BWE superficially looks stronger than EtF the equivalence helps better justify the strong assumptions for constructing EtF.

Definition 6.9 (Blockchain Witness Encryption)**.** *Consider PPT algorithms* (Gen, Enc, Dec) *in the context of a blockchain* Γ^V *is an BWE-scheme with evolved predicate* evolved *and a lottery predicate* lottery *working as follows:*

Setup. $(\mathsf{pv}, \mathsf{td}) \leftarrow \mathsf{Gen}()$ *generates a public value* pv *and an extraction trapdoor* td*. Initially* pv *is put on* $\mathbf{B}$*.*

Encryption. $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, W, m)$ *takes as input a blockchain* $\mathbf{B}$*, including the public value, a PPT function* W*, the witness recogniser, and a message* m*. It outputs a ciphertext* ct*, a blockchain witness encryption.*

Decryption. $m/\perp \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathtt{w})$ *in input a blockchain state* $\tilde{\mathbf{B}}$*, including the a public value* pv*, a ciphertext* ct *a witness* w*, it outputs a message* m *or* $\perp$*.*

Correctness. An BWE-scheme is correct if for honest parties i and j, PPT function W, and witness w such that $W(\mathtt{w}) = 1$ the following holds with overwhelming probability: if party i runs $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathbf{B}, W, m)$ and party j starts running $\mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathtt{w})$ in $\tilde{\mathbf{B}}$ evolved from $\mathbf{B}$, then eventually $\mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathtt{w})$ outputs m.

Security. We establish a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. In section 4.2 we described how $\mathcal{A}$ and $\mathcal{Z}$ execute a blockchain protocol. In addition, we now let the adversary interact with the challenger in a game $\mathsf{Game}^{\text{IND-CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ which can be summarized as follows.

1. $(\mathsf{pv}, \mathsf{td}) \leftarrow \mathsf{Gen}()$ and put pv on the blockchain.
2. $\mathcal{A}$ executes the blockchain protocol Γ together with $\mathcal{Z}$ and at some round r chooses a blockchain $\mathbf{B}$, a function W and two messages m_0 and m_1 and sends it all to $\mathcal{C}$.
3. $\mathcal{C}$ chooses a random bit b and encrypts the message m_b with the parameters it received and sends ct to $\mathcal{A}$.
4. $\mathcal{A}$ continues to execute the blockchain until some round $\tilde{r}$ where the blockchain $\tilde{\mathbf{B}}$ is obtained and the $\mathcal{A}$ outputs a bit b'.

The adversary wins the game if it succeeds in guessing b with probability notably greater than one half without $W(\mathsf{Extract}(\mathsf{td}, \tilde{\mathbf{B}}, \mathsf{ct}, W)) = 1$.

6.7.0.1 EtF from BWE.

We first show the trivial direction of getting EtF from BWE. Let $\Pi_{\mathsf{BWE}} = (\mathsf{Gen}_{\mathsf{BWE}}, \mathsf{Enc}_{\mathsf{BWE}}, \mathsf{Dec}_{\mathsf{BWE}})$ be an BWE scheme. Recall that one wins the lottery if $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 1$. We construct a EtF scheme. To encrypt, let W be the function $W(\mathtt{w}) = \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathtt{w})$ and output $\mathsf{Enc}_{\mathsf{BWE}}(\mathbf{B}, W, m)$. If winning the lottery for $(\mathsf{sl}, \mathsf{R})$ then let $\mathtt{w}$ be the secret key winning the lottery and output $\mathsf{Dec}(\tilde{\mathbf{B}}, \mathsf{ct}, \mathtt{w})$. The proof is straightforward.

6.7.0.2 BWE from EtF.

We now show how to construct BWE from EtF. Let $(\mathsf{Enc}_{\mathsf{EtF}}, \mathsf{Dec}_{\mathsf{EtF}})$ be an EtF scheme. Assume a blockchain with Turing complete smart contracts which can be programmed to send, receive, and reject stake. Assume furthermore that if a constant fraction of the stake is moved to an account then within a polynomial number of slots it will begin winning the lottery with constant probability.

We assume that the contract C of an account is hardcoded into the account when created and cannot be changed. We also need to assume that the blockchain reaches all slot numbers such that there is an independent chance to win at all slot numbers. We also need that only polynomially many slot numbers are reached in polynomial time. We need that the lottery can be filtered such that only certain accounts can win a given role. We need that the filtering can depend on the smart contract put on the account when the account was created.

The construction needs a notion of labelled simulation-sound NIZK proof of knowledge. For such a scheme there is a label connected to a proof and a proof of instance $\mathtt{x}$ and label L cannot be mauled into a proof of instance $\mathtt{x}$ and label $L' \neq L$. This can generically be constructed from an unlabelled scheme simply by letting the label be part of the instance. Let pv of the BWE scheme be the CRS of the NIZK and let td be the extraction trapdoor of the BWE scheme.

To encrypt proceed as follows.

1. Create a fresh account vk with a smart contract E and with no stake on it. Program E with W hard-coded and such that E is willing to receive calls of the form $(\textsc{Transfer}, \pi, f, F)$ from any other smart contract D. If D has f stake available and π is a proof of knowledge of $\mathtt{w}$ such that $W(\mathtt{w}) = 1$ and with label F, then accept a transfer of f stake from D and send them to F.

2. Let filter be the filter which only accepts accounts which have no stake initially and which have smart contracts C of the form that it will only accept stake from the account vk created by the encryptor above.

3. Use $\mathsf{Enc}_{\mathsf{EtF}}$ to encrypt to roles E at slots $2^i + j$ for $i = 1, \ldots, \kappa$ and $j = 1, \ldots, \kappa$. Use the filter filter.

To decrypt create a new account F with a contract accepted by filter. Then use $\mathtt{w}$ to transfer stake to F via E. Note that F is allowed to win the lotteries used in the EtF encryptions. No matter when the decryption is performed, the slots of the blockchain will eventually reach the next slot of the form 2^i as at most polynomially many slots were reached already. After this comes κ slots in a row to which the encryptor encrypted using EtF. Each of these is won with a constant probability. Therefore the probability of not decrypting is negligible.

6.8 ECW from [DS15]

While general constructions of witness encryption (WE) [GGH13a, GGH+13b, GLW14] are impractical, Derler and Slamanig [DS15] puts forward a notion inspired by the standard definition of WE, but weakened by having one extra round. A standard WE scheme consists of two algorithms Enc and Dec (ignoring the setup phase), wherein a user, in a single flow, can encrypt a message m under a specific statement $\mathtt{x}$ and produce a ciphertext ct. A recipient of ct is then able to recover the message if they know a witness $\mathtt{w}$ which certifies the truth of $\mathtt{x}$. The weakened variant of WE in [DS15] is associated with a proof system $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ and consists of two rounds: in the first round, a recipient computes and broadcasts $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w})$. Later, a user can verify the proof and encrypts a message m under $(\mathtt{x}, \pi)$ if $\mathsf{Verify}(\mathsf{crs}, \mathtt{x}, \pi) = 1$.

In this section, we show how to realize ECW from [DS15] and provide additional details on their constructions for the sake of completeness.

6.8.0.1 ECW with respect to Groth-Sahai NIZK Proofs.

Groth-Sahai (GS) proofs work by using dual-mode commitments that are homomorphic with respect to group operations and consist of two setup algorithms. If the commitment parameters are generated by the first algorithm, one obtains perfectly binding commitments. In contrast, the second algorithm generates the parameters in a way that leads to perfectly hiding commitments. The GS protocol aims at convincing a verifier that a set of equations are satisfied by the values inside the commitments. The prover gives to the verifier a proof containing commitments to the witness together with some additional group elements. Given such a proof and based on the linearity of GS commitments, one can encrypt a message with respect to the commitments used within the proof using a *smooth projective hash function* (SPHF) for linear languages (see Section 5.6).

Let us give more details. The commitment parameters in the perfectly binding mode [3] are group elements $[U_1], [U_2], [U_3] \in \mathbb{G}^3 \times \mathbb{G}^3 \times \mathbb{G}^3$ defined as follows:

$$[U_1] = ([\tau_1], [0], [1]) \quad ; \quad [U_2] = ([0], [\tau_2], [1])$$
$$[U_3] = \tau_3 \cdot [U_1] + \tau_4 \cdot [U_2] = ([\tau_1\tau_3], [\tau_2\tau_4], [\tau_3 + \tau_4])$$

[3] Here we only focus on the perfectly binding mode. All the explanations can also be applied to the perfectly hiding mode in a straightforward manner.

where $\tau_1, \tau_2, \tau_3, \tau_4 \leftarrow\$ \mathbb{Z}_q$. Setting these parameters, a commitment to a message $m \in \mathbb{G}$ is by choosing $r_1, r_2, r_3 \leftarrow\$ \mathbb{Z}_q$ and computing

$$\begin{aligned}\mathsf{cm}_m &= ([0],[0],[m]) + r_1 U_1 + r_2 U_2 + r_3 U_3 \\ &= ([\tau_1(r_1 + \tau_3 r_3)], [\tau_2(r_2 + \tau_4 r_3)], [m + r_1 + r_2 + r_3(\tau_3 + \tau_4)]).\end{aligned}$$

Now let $\mathtt{lpar} = ([U_1],[U_2],[U_3])$ and define

$$\mathcal{L}_{\mathtt{lpar}} = \{([m], \mathsf{cm}_m) | \exists (r_1, r_2, r_3) \in \mathbb{Z}_q^3 : \mathsf{cm}_m = ([0],[0],[m]) + r_1 U_1 + r_2 U_2 + r_3 U_3\}$$

It is not hard to see that $\mathcal{L}_{\mathtt{lpar}}$ is a linear language and therefore a SPHF for it can be constructed as follows:

setup(1^λ): Run the bilinear group generation algorithm and let p be the bilinear group description. Also, set the language parameters $\mathtt{lpar} = ([U_1],[U_2],[U_3])$ as defined above.

hashkg($\mathtt{lpar}$): Choose $\alpha_1, \alpha_2, \alpha_3 \leftarrow\$ \mathbb{Z}_q^3$ and return $\mathsf{hk} = (\alpha_1, \alpha_2, \alpha_3)$.

projkg($\mathtt{lpar}; \mathsf{hk}, \mathtt{x}$): Parse $\mathtt{lpar}$ as $([U_1],[U_2],[U_3])$ and hk as $(\alpha_1, \alpha_2, \alpha_3)$. Return $\mathsf{hp} = (\gamma_1, \gamma_2, \gamma_3) \in \mathbb{G}^3$, where

$$\begin{aligned}\gamma_1 &= \alpha_1[\tau_1] + [\alpha_3];\ \gamma_2 = \alpha_2[\tau_2] + [\alpha_3] \\ \gamma_3 &= \alpha_1[\tau_1\tau_3] + \alpha_2[\tau_2\tau_4] + \alpha_3[\tau_3 + \tau_4]\end{aligned}$$

hash($\mathtt{lpar}; \mathsf{hk}, \mathtt{x}$): Parse the statement $\mathtt{x}$ as $([m], \mathsf{cm}_m = ([u],[v],[e]))$, and hk as $(\alpha_1, \alpha_2, \alpha_3)$. Return H computed as $\mathsf{H} = [u] \cdot \alpha_1 + [v] \cdot \alpha_2 + ([e] - [m]) \cdot \alpha_3$.

projhash($\mathtt{lpar}; \mathsf{hp}, \mathtt{x}, \mathtt{w}$): Parse hp as $(\gamma_1, \gamma_2, \gamma_3)$ and $\mathtt{w}$ as (r_1, r_2, r_3). Return pH computed as $\mathsf{pH} = \gamma_1 r_1 + \gamma_2 r_2 + \gamma_3 r_3$.

We refer the reader to [DS15] for the security proof of the above SPHF.

Equipped with this construction, one can now construct an ECW by encoding the lottery statement into vectors of commitments satisfying pairing-product equations (PPEs). In more details, the construction works as follows. All receivers who have $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$ publish a GS proof π_i that they have such secret key $\mathsf{sk}_{L,i}$. The encrypting party encodes the lottery predicate into a set of pairing-product equations (PPEs) and encrypts the message under each of these GS proofs using the SPHF scheme described above. We note that this construction can be used for any type of algebraic lottery that can be represented as a set of pairing product equation (e.g., algebraic VRF-based lotteries). Moreover, while this can be seen as a weaker variant of ECW where the (claimed) winners are required to send a proof of winning the lottery in advance and thus requires an extra round of communication, it results in a construction with significant improvement on the ciphertext size published by the encrypting party, i.e., only linear in the number of winners receiving the message.

6.9 Constructions of cWE

6.9.1 cWE from MS-NISC

Two-Party Non-Interactive Secure Computation (2P-NISC) is a type of protocol where a sender S (with input y) and a receiver R (with input x) want to jointly compute a function $f(x, y)$. In this setting, R first publishes a first message. Then, any sender S holding an input y can send a message to the receiver revealing only $f(x, y)$ to R. As usual in secure computation, the protocol must provide privacy of the inputs and correctness of the output.

Afshar et al. [AMPR14] introduced another flavour of NISC called *Multi-Sender NISC* (MS-NISC) where the first message can be reused to run secure computation with many different senders. That is, R, with input x, publishes a first message as before, but now any party who wants to participate in secure computation with R can send back a message to S who can then output the result of the computation. The ideal functionality of MS-NISC as presented in [AMPR14] is depicted in Fig. 6.2.

MS-NISC Functionality

Assume $f(\perp, \cdot) = f(\cdot, \perp) = \perp$.

- Initialize a list, L, of pairs of strings.
- Upon receiving a message $(\texttt{input}, x)$ from R, store x and continue.
  1. Upon receiving message $(\texttt{input}, y)$ from S_i, insert the pair (S_i, y) into L. If R is corrupted send $(S_i, f(x, y))$ to the adversary. Otherwise, send $(\texttt{messageReceived}, S_i)$ to R.
  2. Upon receiving a message `getOutputs` from R, send $\{(S_i, f(x, y))\}_{(S_i,y)\in L}$ to R.

Figure 6.2: MS-NISC Functionality $\mathcal{F}_{\text{MS-NISC}}$

In Fig. 6.3, we show how to construct cWE by having black-box access to $\mathcal{F}_{\text{MS-NISC}}$. The main idea is that a party acts as a receiver and sends the first message in MS-NISC containing its witness $\mathtt{w}$ in order to provide a "commitment" to that witness. Later on, any other party can use this "commitment" to create a cWE ciphertext by sending an encryption of the message and acting as the sender of the MS-NISC to provide a second message that allows for evaluating a function $f(\mathtt{w}, y)$ that outputs a decryption key iff the witness $\mathtt{w}$ satisfies a given relation.

Note that the ideal functionalities used in the construction are stated for clarity and is not compatible with our game-based notion of security for cWE. By assuming a concrete secure realization of the above functionalities, one can argue about security using the corresponding simulator and use that to extract witnesses from commitments and make the proof go through.

Construction of cWE based on MS-NISC

Initialization: Initialize $\mathcal{F}_{\text{MS-NISC}}$ by instantiating a list L of pairs of strings.

Commit: R proceeds as follows:

- Commits to its witness $\mathtt{w}$ by calling $\mathcal{F}_{\text{MS-NISC}}$ on input $(\mathtt{input}, \mathtt{w})$.

Encryption: S proceeds as follows:

- Generates a key k of length $|m|$ and encrypts the message m as $\mathsf{ct} \leftarrow k \oplus m$.
- Calls $\mathcal{F}_{\text{MS-NISC}}$ on input $(\mathtt{x}, k)$ and sends ct directly to R.

Decryption: R receives $(\mathtt{messageReceived}, S)$ from $\mathcal{F}_{\text{MS-NISC}}$ and ct from S and proceeds as follows:

- Calls $\mathcal{F}_{\text{MS-NISC}}$ on input $\mathtt{getOutputs}$.
- Upon receiving k from $\mathcal{F}_{\text{MS-NISC}}$, outputs $m \leftarrow k \oplus \mathsf{ct}$.

Figure 6.3: Construction of cWE based on MS-NISC

6.9.2 cWE using Garbled Circuits and Oblivious Transfer

Instead of relying on the full MS-NISC functionality in a black-box way, we now do a careful analysis resulting in a protocol which uses only the properties of MS-NISC needed to obtain a protocol that satisfies the definition of cWE.

We observe that the correctness property in the definition of cWE only requires that a correctly generated ciphertext can be decrypted by the decryption algorithm. Thus, we expect the second message of MS-NISC functionality to be generated correctly. In particular, when looking into the internals of the protocol in [AMPR14], we observe that we can construct cWE from a MS-NISC protocol without the precautions against a malicious sender S. However, we still want to make sure that we preserve authenticity of the underlying garbled circuit scheme. This property guarantees that no garbled output can be constructed different from what is dictated by the function and its inputs. In other words, the only thing a malicious receiver can do with the garbled circuit is to evaluate it on the committed input. Finally, we observe that privacy of input is not a requirement for the sender. Thus, we can consider variants of garbled circuit schemes without privacy guarantees.

Privacy-free Garbled Circuits. One of the most efficient GC schemes in terms of communication is the scheme by [ZRE15] based on a technique called half-gates. Using their technique in the privacy-free setting results in garbled circuits containing one ciphertext for each AND gate and no ciphertexts for XOR gates.

cWE from privacy-free GC and OT. We conclude this section by presenting an efficient construction of cWE using only a privacy-free garbled circuit and oblivious transfer. To this end, let us first recall some required definitions.

Let $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode}, \mathsf{Verify})$ be a garbled circuit with correctness and authenticity, and $\Pi_{\mathsf{OT}} = (\Pi^R_{\mathsf{OT}}, \Pi^S_{\mathsf{OT}}, \Pi^O_{\mathsf{OT}})$ be an oblivious transfer protocol that realizes $\mathcal{F}_{\mathsf{OT}}$. We consider two parties E and D that respectively play the role of the encryptor and the decryptor in an execution of the cWE scheme. The construction of $\Pi_{\mathsf{cWE}} = (\mathsf{Enc}, \mathsf{Dec})$ with commitment Π^R_{OT} for circuit class $\mathcal{C}$ is given in Fig. 6.4.

CWE from OT and GC

Primitives: A correct and authentic garbling scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Encode}, \mathsf{Eval}, \mathsf{Decode})$, and a 2-round OT $\Pi_{\mathsf{OT}} = (\Pi^R_{\mathsf{OT}}, \Pi^S_{\mathsf{OT}}, \Pi^O_{\mathsf{OT}})$.
Commit: D with secret $\mathsf{s} \in \{0,1\}^n$ plays the role of the receiver in n instances of Π_{OT} and computes $(\mathsf{cm}, \mathtt{w})$ as follows:

- Select $\rho^R_j \leftarrow\$ \{0,1\}^\lambda$, and compute $m^R_j \leftarrow \Pi^R_{\mathsf{OT}}(\mathsf{s}_j; \rho^R_j)$ for $j \in [n]$.
- Define $\mathsf{cm} = \{m^R_j\}_{j\in[n]}$, and $\mathtt{w} = (\mathsf{s}, \{\rho^R_j\}_{j\in[n]})$. Note that $\mathtt{w}$ can be seen as an opening of cm.

Common inputs: A security parameter λ, a circuit $C \in \mathcal{C}$, a commitment key ck, and a statement $\mathtt{x} = (\mathsf{cm}, C, y)$.
Encryption.: E plays the role of the sender in n instances of Π_{OT} and computes a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ as follows:

1. Let $C_{\mathtt{x}}$ be a circuit that realizes the following relation $\mathcal{R}$ on $\mathtt{x}$: $\mathcal{R}(\mathtt{x} = (\mathsf{cm}, C, y), (\mathsf{s}, \vec{\mathsf{d}})) = 1$ iff $(\mathsf{s}, \vec{\mathsf{d}})$ opens cm and $C(\mathsf{s}) = y$. Compute $(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C_{\mathtt{x}})$, where $e := \{k^0_j, k^1_j\}_{j\in[n]}$, and $d := (k^0, k^1) \in \{0,1\}^{2|m|}$.
2. For $j \in [n]$, select $\rho^S_j \leftarrow\$ \{0,1\}^\lambda$, and compute $m^S_j = \Pi^S_{\mathsf{OT}}(k^0_j, k^1_j, m^R_j; \rho^S_j)$.
3. Compute $\mathsf{ct}_1 = k^1 \oplus m$ and $\mathsf{ct}_2 = (\mathbf{C}, \{m^S_j\}_{j\in[n]})$.
4. Send $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ to D.

Decryption: Given $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and $\mathtt{w} = (\mathsf{s}, \{\rho^R_j\}_{j\in[n]})$, D proceeds as follows:

1. Parse ct_2 as $(\mathbf{C}, \{m^S_j\}_{j\in[n]})$.
2. Execute $k^{\mathsf{s}_j}_j = \Pi^O_{\mathsf{OT}}(m^S_j, \mathsf{s}_j, \rho^R_j)$ for $j \in [n]$, and $Y = \mathsf{Eval}(\mathbf{C}, \{k^{\mathsf{s}_j}_j\}_{j\in[n]})$.
3. Compute $m = Y \oplus \mathsf{ct}_1$.

Figure 6.4: cWE based on GC and OT

Theorem 6.3. *Let $\mathcal{C}$ be a class of circuits. Let Π_{OT} be an OT protocol that realizes $\mathcal{F}_{\mathsf{OT}}$ and GC be a correct and authentic garbling scheme. The cWE scheme Π_{cWE} for $\mathcal{C}$ in Fig. 6.4 is correct and semantically secure as defined in Definition 6.3.*

Proof. **(Correctness).** Follows directly from the correctness property of the Π_{OT} and GC. **(Semantic Security).** Assume that $\mathcal{A}$ is a PPT adversary against semantic security of Π_{cWE} such that, for adversarially chosen values $(\mathsf{s}, \rho, C, y, m_0, m_1)$, given an encryption of m_b under statement $\mathtt{x} = (\mathsf{cm}, C, y)$, where $\mathsf{cm} = \mathsf{Commit}(\mathsf{s}; \rho)$ and $y \neq C(\mathsf{s})$, $\mathcal{A}$ can guess the bit b with non-negligible advantage. We first observe that by the construction of Π_{cWE}, $\mathcal{A}$ can guess b correctly only by distinguishing the correct label k^1 from random. Informally, given that $C(\mathsf{s}) \neq y$, there are only two possible cases in which $\mathcal{A}$ can distinguish k^1 from random: either by the ability to gain knowledge about invalid labels $k^{1-\mathsf{s}_j}_j$ that do not correspond to $\mathcal{A}$'s committed value, or by the ability to gain knowledge about k^1 directly.

We show that a successful adversary in the first case can be used to break the sender security of Π_{OT} whereas a successful adversary in the second case can be exploited to break the the authenticity of GC.

In order to formally prove semantic security, we first define the experiment $\mathsf{Exp}^{\mathsf{SS-b}}_{\mathcal{A},\lambda}$ in Algorithm 13. We define b' as the output of $\mathsf{Exp}^{\mathsf{SS-b}}_{\mathcal{A},\lambda}$. Note that $\mathsf{Exp}^{\mathsf{SS-b}}_{\mathcal{A},\lambda}$ corresponds to the

Algorithm 13: $\mathsf{Exp}^{\mathsf{SS-b}}_{\mathcal{A},\lambda}$

$(\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)$
$\rho = \rho_1 || \ldots || \rho_n$; $\mathsf{cm} = \{m_j^R\}_{j\in[n]}$, where $m_j^R \leftarrow \Pi_{\mathsf{OT}}^R(\mathsf{s}_j; \rho_j)\ \forall j \in [n]$.
$\mathsf{x} := (\mathsf{cm}, C, y)$; $(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C_{\mathsf{x}})$ where $e := \{k_j^0, k_j^1\}_{j\in[n]}$, and $d := (k^0, k^1)$.
$\rho_j^S \leftarrow\$ \{0,1\}^\lambda$; $m_j^S = \Pi_{\mathsf{OT}}^S(k_j^0, k_j^1, m_j^R; \rho_j^S)$ $(\forall j \in [n])$.
$\mathsf{ct}_1 = k^1 \oplus m_b$ and $\mathsf{ct}_2 = (\mathbf{C}, \{m_j^S\}_{j\in[n]})$; $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$.
$\mathsf{ct} := \perp$ if $C(\mathsf{s}) = y$ or $C \notin \mathcal{C}$ or $|m_0| \neq |m_1|$
$b' \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{ct})$

semantic security experiment of Π_{cWE} in Definition 6.3, except that b is fixed.

To prove the theorem, let us assume by contradiction that there is an adversary $\mathcal{A}$ that breaks the semantic security of Π_{cWE}. That is, for a non-negligible function ϵ, we have

$$\left|\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{SS-0}}_{\mathcal{A},\lambda}] - \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{SS-1}}_{\mathcal{A},\lambda}]\right| \geq \epsilon(\lambda)$$

We now use a standard hybrid argument and define several games, where the first is $\mathsf{Exp}^{\mathsf{SS-0}}_{\mathcal{A},\lambda}$, the last is $\mathsf{Exp}^{\mathsf{SS-1}}_{\mathcal{A},\lambda}$, and the intermediate hybrids are defined as follows:
Hybrid 0 is defined as $\mathsf{Exp}^{\mathsf{SS-0}}_{\mathcal{A},\lambda}$.

Hybrid 1 is the same as **Hybrid 0**, except that the messages $\{m_j^S\}_{j\in[n]}$ are computed by the OT simulator i.e., as $\{m_j^S\}_{j\in[n]} \leftarrow \mathsf{Sim}(1^\lambda, \{m_j^R\}_{j\in[n]})$.

Hybrid 2 is the same as **Hybrid 1**, except that ct_1 is defined as $\mathsf{ct}_1 := k^1 \oplus m_1$.

Hybrid 3 is defined as $\mathsf{Exp}^{\mathsf{SS-1}}_{\mathcal{A},\lambda}$.

By assumption, $\mathcal{A}$ must distinguish some pair of adjacent intermediate hybrids. That is, for some $i \in \{0,1,2\}$, we must have

$$\left|\Pr[1 \leftarrow \mathsf{Hybrid}^i_{\mathcal{A},\lambda} - \Pr[1 \leftarrow \mathsf{Hybrid}^{i+1}_{\mathcal{A},\lambda}]\right| \geq \frac{1}{3}\epsilon(\lambda)$$

We now analyze all three cases:

- $i = 0$. Notice that the only difference between **Hybrid 0** and **Hybrid 1** is that in the former, the sender's message $\{m_j^S\}_{j\in[n]}$ is computed by a real sender (World 0), whereas in the latter, it is computed by the simulator (World 1). Assuming that $\mathcal{A}$ can distinguish hybrids 0 and 1, we construct an adversary $\mathcal{B}$ against sender security of Π_{OT} that distinguishes the two worlds with the same probability. $\mathcal{B}$ works as follows:

1. $\mathcal{B}$ invokes $\mathcal{A}(1^\lambda)$ and obtains $(\mathsf{s}, \rho, C, y, m_0, m_1)$.

2. If $|m_0| \neq |m_1|$ or $C(\mathsf{s}) = y$, $\mathcal{B}$ aborts; otherwise, it parses $\rho = \rho_1 || \ldots || \rho_n$ and defines $\mathsf{cm} = \{m_j^R\}_{j\in[n]}$, where $m_j^R \leftarrow \Pi_{\mathsf{OT}}^R(\mathsf{s}_j; \rho_j)$ for $j \in [n]$. Let $\mathtt{x} = (\mathsf{cm}, C, y)$. As an environment controlling the OT execution, $\mathcal{B}$ provides the input of sender and receiver to the OT challenger as follows:

   - computes a garbling of circuit $C_{\mathtt{x}}$ (as defined in Fig. 6.4) by $(\mathbf{C}, e, d) \leftarrow \mathsf{Garble}(1^\lambda, C_{\mathtt{x}})$ and sends the input keys to the OT challenger as the sender's input.

   - sends s to the OT challenger as the receiver's choice bits.

3. The OT challenger computes the sender's message $\{m_j^S\}_{j\in[n]}$ either by invoking a real sender (World 0), or by invoking the simulator (World 1), and sends it to $\mathcal{B}$.

4. $\mathcal{B}$ parses $d = (k^0, k^1)$, and forwards $\mathsf{ct} := (\mathsf{ct}_1, \mathsf{ct}_2)$ to the cWE adversary $\mathcal{A}$, where $\mathsf{ct}_1 = k^1 \oplus m_0$ and $\mathsf{ct}_2 = (\mathbf{C}, \{m_j^S\}_{j\in[n]})$.

It is clear that $\mathcal{B}$ has the same advantage in breaking sender security of Π_{OT} as $\mathcal{A}$ in distinguishing the two hybrids.

- $i = 1$. The only difference in **Hybrid 1** and **Hybrid 2** is in how we generate ct_1 (that is $\mathsf{ct}_1 := k^1 \oplus m_{i-1}$ in **Hybrid i**). To argue indistinguishability of the two hybrids, it suffices to show that k^1 is indistinguishable from random. To achieve this, we observe that because in both hybrids, the sender's message $\{m_j^S\}_{j\in[n]}$ is computed by the simulator i.e., as $\{m_j^S\}_{j\in[n]} \leftarrow \mathsf{Sim}(1^\lambda, \{m_j^R\}_{j\in[n]})$, $\mathcal{A}$ cannot distinguish k^1 from random by the ability of knowing invalid labels. Thus, the only way $\mathcal{A}$ can distinguish k^1 from random should be by directly forging an output key k^1 for the garbled circuit $\mathbf{C}$. It is therefore straightforward to use a successful adversary that distinguishes the two hybrids with non-negligible advantage to break the authenticity of the underlying garbling scheme.

- $i = 2$. This is handled identically to $i = 0$, except that in this case $\mathsf{ct}_1 := k^1 \oplus m_1$ encrypts m_1 instead of m_0.

We conclude the proof by this observation that in any of the three cases, we reach a contradiction and thus our assumption of the existence of $\mathcal{A}$ against the semantic security of Π_{cWE} cannot be true. □

Remark 6.3. *The commitment scheme in Π_{cWE} is the receiver's algorithm of Π_{OT} and therefore by UC-security of Π_{OT}, it satisfies both extractability and hiding property. Using Lemma 6.1 and weak semantic security shown in Theorem 6.3, one can then conclude that Π_{cWE} also achieves strong semantic security.*

6.10 Proof of Security for Our EtF Construction

Here we prove security of Theorem 6.2. Recall we proceed in two steps: first we consider an idealized case where there is no threshold committee; we then show we can prove security of our threshold construction from this setting.

CCA and resharing. Our proof below ignores decryption oracles for the IBE scheme and considers the case where the master secret key is shared only at the beginning (instead of at each slot). We observe these can be accounted for with additional hybrids in our proof below in a straightforward manner.

6.10.1 The non-threshold case.

The simplified setting we will now show security for is in Fig. 6.5.

Non-threshold EtF

The non-threshold setting we consider is the same as that in Section 6.6.1 with the following exceptions:

- At the beginning of the run of the blockchain, there is no sharing of the master secret key of the IBE scheme.
- We let the honest parties run exactly as in the other construction, with the exception that they validate and messages related to the shares of the master secret keys, as well as of the secret keys for specific slots.
- We change the way we encapsulate the secret-key for a certain slot. While in Section 6.6.1 we require committee members to each broadcast a ciphertext containing a share of the secret-key for slot sl, here we instead replace that stage with the execution of the following oracle $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$.

  $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}(\mathbf{B}, \mathsf{sl})$:
  - $\mathsf{sk}_{\mathsf{sl}} \leftarrow \mathsf{IBE.Keygen}(\mathsf{msk}, (\mathsf{sl}, R))$
  - $\mathsf{ct}_{\mathsf{sl}}^{\mathsf{sk}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, R, \mathsf{sk}_{\mathsf{sl}})$
  - Broadcast $\mathsf{ct}_{\mathsf{sl}}^{\mathsf{sk}}$

Figure 6.5: Hybrid non-threshold setting for proof of security

A point on the view of the adversary: we recall that, at any given point in time, a valid blockchain execution contains ciphertexts $\mathsf{ct}_{\mathsf{sl}}^{\mathsf{sk}}$, encrypting slot-specific secret keys for the winner of the slot sl in the chain. In the non-threshold setting, they correspond to the output of the key-provider oracle (in the actual construction, there are more ciphertexts, each containing a share of the key).

Now assume an adversary $\mathcal{A}_{EtF}^{\text{no-thresh}}$ for the EtF security experiment controlling at most an α fraction of the stake with non-negligible success probability in the EtF security experiment. We first to construct an adversary $\mathcal{A}_{\mathsf{IBE}}$ for IBE security using $\mathcal{A}_{EtF}^{\text{no-thresh}}$. Adversary $\mathcal{A}_{\mathsf{IBE}}$ works as follows:

- On receiving the IBE public parameters from the IBE challenger, it injects into blockchain genesis block the IBE's master public. The adversary $\mathcal{A}_{EtF}^{\text{no-thresh}}$ declares a corrupted set of parties S_{corr} and then $\mathcal{A}_{\text{IBE}}$ runs an execution of the blockchain with $\mathcal{A}_{EtF}^{\text{no-thresh}}$ where $\mathcal{A}_{\text{IBE}}$ simulates the honest parties. In this execution $\mathcal{A}_{\text{IBE}}$ acts as key-provider oracle, which it emulates as follows. We distinguish two cases depending on whether the winner of the slot is a corrupted party or an honest one[4]. On query $(\mathbf{B}, \mathsf{sl})$:

  - *if a corrupted party has won* the role for slot sl (i.e. $\mathsf{winners}(\mathbf{B}, \mathsf{sl}, R) \cap S_{\text{corr}} \neq \emptyset$) then invoke the IBE challenger oracle on identity sl obtaining $\mathsf{sk}_{\mathsf{sl}}$ and broadcast $\mathsf{ct}_{\mathsf{sl}}^{\mathsf{sk}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, R, \mathsf{sk}_{\mathsf{sl}})$.

  - *if a corrupted party has not won* the role for slot sl then broadcast the encryption of a dummy plaintext $\mathsf{ct}_{\mathsf{sl}}^{\mathsf{sk}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, R, \vec{0})$ where $\vec{0}$ is a string of zeros of the appropriate length.

  The intuitive reason for separating the two cases is that we want to query the same slots that $\mathcal{A}_{EtF}^{\text{no-thresh}}$ wins and no more. In particular we do not want to query the challenge slot sl^* (defined next). Notice, in fact, that only the slots for which the adversary has a corrupted winner will be asked to the IBE key-generation oracle. At the end of this stage, $\mathcal{A}_{EtF}^{\text{no-thresh}}$ will return $(\mathbf{B}, \mathsf{sl}^*, R, m_0, m_1)$ and $\mathcal{A}_{\text{IBE}}$ will forward $((\mathsf{sl}^*, R), m_0, m_1)$ to the IBE challenger.

- After receiving a ciphertext ct^* from the IBE challenger, $\mathcal{A}_{\text{IBE}}$ forwards it to $\mathcal{A}_{EtF}^{\text{no-thresh}}$. Then $\mathcal{A}_{\text{IBE}}$ simulates the execution of the blockchain as described above. At the end of the execution $\mathcal{A}_{EtF}^{\text{no-thresh}}$ outputs a guessing bit b^* which $\mathcal{A}_{\text{IBE}}$ forwards to the IBE challenger.

We claim that the advantage of $\mathcal{A}_{\text{IBE}}$ in the IBE experiment is negligibly close to that of $\mathcal{A}_{EtF}^{\text{no-thresh}}$ in the EtF non-threshold experiment (the one without threshold sharing). With that goal in mind, we first show that the inputs we feed to $\mathcal{A}_{EtF}^{\text{no-thresh}}$ and the blockchain execution emulated by $\mathcal{A}_{\text{IBE}}$ is indistinguishable from that in the EtF experiment. Notice that the only difference in the distributions is in the ciphertexts for the non-corrupted winners. If we could distinguish between the two cases, then we could break security of the ECW scheme. Therefore the views of $\mathcal{A}_{EtF}^{\text{no-thresh}}$ in the two cases is indistinguishable. Finally, we lower-bound the success probability of $\mathcal{A}_{\text{IBE}}$. Intuitively, we can observe that two adversaries return the same experiment bit. The only aspect that could impair $\mathcal{A}_{\text{IBE}}$'s success probability compared to $\mathcal{A}_{EtF}^{\text{no-thresh}}$'s is the possibility of having asked the IBE key-generation oracle for the challenge slot sl^*. We observe this does not affect the success probability of $\mathcal{A}_{\text{IBE}}$. Formal details are in Section 6.10.3.

6.10.2 Security of threshold construction from non-threshold case.

The argument above had a simplified setting where we abstracted out all the threshold aspects of the protocol. This includes the committee holding shares of the master secret key and dealing shares of the slot-specific secret key. We now prove security for the

[4] Notice that we can check this for both types of parties as discussed in Section 4.4.2.

actual threshold scenario (Section 6.6.1) building an adversary for our actual (threshold) construction using the adversary for the non-threshold construction (Fig. 6.5).

The threshold adversary $\mathcal{A}_{EtF}^{\text{thresh}}$ needs to emulate the setting for the other adversary where there is a single ciphertexts containing the slot-specific secret key (instead of several containing their shares). It works as follows. First, it corrupts the same parties as $\mathcal{A}_{EtF}^{\text{no-thresh}}$ and executes a blockchain as $\mathcal{A}_{EtF}^{\text{no-thresh}}$ does and broadcasting the same messages it does, with one exception which we now describe. The views of two adversaries (threshold vs non-threshold) differ in only one respect—and so do the two respective blockchains executions. The view of the threshold execution contains ciphertexts of this type for each winning slot sl (we use bracket notation for shares for readability): $\left(\left(\mathsf{ct}_{\mathsf{sl}}^{\text{hon}}[j]\right)_{j\notin S_{\text{corr}}}, \left(\mathsf{ct}_{\mathsf{sl}}^{\text{cor}}[j]\right)_{j\in S_{\text{corr}}}\right)$ These contain the shares for the slot-specific slot sl. The view for the non-threshold execution instead contains a single ciphertext with slot-specific secret key. For a honest slot not corrupted by the adversary, we denote it by $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{hon}}$, otherwise we denote it by $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{cor}}$. During the blockchain execution $\mathcal{A}_{EtF}^{\text{no-thresh}}$ will expect to see some ciphertext ($\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{hon}}/\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{cor}}$) whenever a slot is won, which corresponds to a query of $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$. The threshold adversary $\mathcal{A}_{EtF}^{\text{thresh}}$ can emulate this as follows. For every query to $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$:

- *if the slot is won by a honest party*, then broadcast $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{hon}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, \vec{0})$ for a vector of zeros of the appropriate length.

- *if the slot is won by a corrupted party*, then its view will contain $\left(\mathsf{ct}_{\mathsf{sl}}^{\text{cor}}[j]\right)_{j\in[n]}$. It can then decrypt them, combine the obtained shares into a slot key $\mathsf{sk}_{\mathsf{sl}}$ and broadcast $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{cor}} \leftarrow \mathsf{Enc}_{\mathsf{ECW}}(\mathbf{B}, \mathsf{sl}, \mathsf{sk}_{\mathsf{sl}})$

After receiving challenge messages from $\mathcal{A}_{EtF}^{\text{no-thresh}}$, adversary $\mathcal{A}_{EtF}^{\text{thresh}}$ simply forwards them to its challenger, then continues the execution as above. Finally it outputs the same output guess as $\mathcal{A}_{EtF}^{\text{no-thresh}}$.

We now claim that a successful non-threshold adversary $\mathcal{A}_{EtF}^{\text{no-thresh}}$ for the construction in *Fig*. 6.5 would allow $\mathcal{A}_{EtF}^{\text{thresh}}$ to have a similar advantage (up to negligible additive factors). We proceed by a standard hybrid argument. We define the first hybrid H_0 as the output of running the $\mathcal{A}_{EtF}^{\text{thresh}}$ adversary as just described. The "terminal" hybrid H_6 is defined as the output of running the $\mathcal{A}_{EtF}^{\text{no-thresh}}$ adversary. The intermediate hybrids are as follows.

- H_1: like H_0 except that we change one step in how $\mathcal{A}_{EtF}^{\text{thresh}}$ emulates $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$. Specifically, for the case of the honest parties, we now run $(\mathsf{sk}_i^{\mathsf{ID}})_{i\in[n]} \leftarrow \mathsf{Sim}_{\mathsf{kg}}(\mathsf{mpk}, (\mathsf{msk}_i)_{i\in S_{\text{corr}}}, \mathsf{sl})$ to simulate the shares of the honest parties. This simulator exists by key-generation simulation of the threshold IBE scheme. We can then combine all shares to obtain a slot-specific key, encrypt it through ECW and then broadcast the encryption $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\text{hon}}$. We have that $H_0 \approx H_1$ because of the security of ECW, since otherwise we would be able to distinguish encryptions of zeros from encryptions of the (combination of) the simulated slot-specific key shares.

- H_2: as previous item but now, instead of the actual secret shares, we give $\mathcal{A}_{EtF}^{\text{thresh}}$ produced by $\mathsf{Sim}_{\mathsf{msk}}$, the simulator from master secret key shares simulation of the threshold IBE scheme. $H_1 \approx H_2$ follows by the same property.

- H_3: like the previous hybrid, but now we replace the blockchain execution from H_2 with one where we do not use the shares to produce $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\mathsf{hon}}$ and $\hat{\mathsf{ct}}_{\mathsf{sl}}^{\mathsf{cor}}$. Instead we move to a blockchain execution as in Fig. 6.5 with the difference that $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$ has a master secret key computed as follows. Let msk be the master secret key obtained by combining the (simulated) shares msk_i. Then we just run $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$ with this master secret key every time we need to provide a ciphertext for a new winning slot. We have $H_2 \approx H_3$ by definition of $\mathcal{O}_{\mathsf{msk}}^{\text{k-provider}}$, by correctness of the underlying homomorphic secret sharing scheme and the simulation of key-generation evaluations of IBE.

- H_4: as before but we now define msk not as the combination of the shares, but as the output of $\mathsf{Sim}_{\mathsf{msk}}$ on the master public key and the corruption set. $H_3 \approx H_4$ follows by simulation of the master secret-key property of the threshold IBE.

- H_5: Like previous item but now we do not use the key-generation simulator and instead apply the key-generation of the IBE before providing a ciphertext. $H_4 \approx H_5$ again follows by the key-generation simulation of the threshold IBE scheme. Also this is the same as H_6 by construction.

6.10.3 Bounding the Advantage of $\mathcal{A}_{\mathsf{IBE}}$ in Proof of Theorem 6.2

Here we formally claim that the advantage of $\mathcal{A}_{\mathsf{IBE}}$ in the IBE experiment is negligibly close to that of $\mathcal{A}_{EtF}^{\text{no-thresh}}$ in the EtF non-threshold experiment:

$$\begin{aligned}
\Pr[\mathsf{WinIBE}] &\geq \Pr[\neg\mathsf{QryClgSlot} \ \wedge \ \mathsf{WinEtFHyb}] \\
&= (1 - \Pr[\mathsf{QryClgSlot} \mid \mathsf{WinEtFHyb}]) \cdot \Pr[\mathsf{WinEtFHyb}] \\
&\approx (1 - \Pr[S_{\mathsf{corr}} \cap \mathsf{winners}(\mathsf{sl}^*) \neq \emptyset \mid \mathsf{WinEtFHyb}]) \cdot \Pr[\mathsf{WinEtFHyb}] \\
&= \Pr[\mathsf{WinEtFHyb}]
\end{aligned}$$

Above the $\mathsf{QryClgSlot}$ is the event where $\mathcal{A}_{\mathsf{IBE}}$ queries the challenge slot in the IBE experiment; WinIBE is the event where $\mathcal{A}_{\mathsf{IBE}}$ wins in the IBE experiment; $\mathsf{WinEtFHyb}$ is the event where $\mathcal{A}_{EtF}^{\text{no-thresh}}$ wins in the EtF experiment against the non-threshold hybrid model (Fig. 6.5). The first inequality follows by construction of $\mathcal{A}_{\mathsf{IBE}}$. The following ones follow from elementary probability theory and from observing that $\mathcal{A}_{\mathsf{IBE}}$ could query the challenge slot only if that was among the corrupted set (but this does not occur condition on the success of $\mathcal{A}_{EtF}^{\text{no-thresh}}$ by the definition of EtF security).

6.11 Further Proofs and Formal Details

6.11.1 Strong Semantic Security of cWE

Informally, this property states that encrypting a message m with respect to a false statement $\mathbf{x} = (\mathsf{cm}, C, y)$ produces indistinguishable ciphertexts to an adversary $\mathcal{A}$ who knows the commitment opening, even if $\mathcal{A}$ gets to see encryptions of m under other (possibly true) statements $\mathbf{x}_i = (\mathsf{cm}_i, C, y)$ but with unknown commitment opening. Formally, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, all auxiliary strings aux and all PPT adversaries $\mathcal{A}$:

$$\left| 2 \cdot \Pr\left[\begin{array}{r} \mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda) \\ (\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{com}}(\cdot)}(\mathsf{ck}, \mathsf{aux}) \\ \mathsf{cm} \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho);\ b \leftarrow\$ \{0,1\} \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, \mathbf{x} = (\mathsf{cm}, C, y), m_b) \\ \forall \mathsf{cm}_i \in \mathcal{Q} : \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ck}, \mathbf{x}_i = (\mathsf{cm}_i, C, y), m_b) \\ \mathbf{ct} := \{\mathsf{ct}\} \cup \{\mathsf{ct}_i\}_{i \in [|\mathcal{Q}|]} \\ \mathbf{ct} := \bot \text{ if } C(\mathsf{s}) = y \text{ or } C \notin \mathcal{C} \end{array} : \mathcal{A}(\mathsf{st}, \mathbf{ct}) = b \right] - 1 \right| \leq \mu(\lambda)$$

where $\mathcal{O}_{\mathsf{com}}(\cdot)$ is a commitment oracle parametrized by ck and defined as follows: on input s_i, computes and returns $\mathsf{cm}_i \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}_i; \rho_i)$ for some randomness ρ_i, and stores cm_i in $\mathcal{Q}$.

Lemma 6.1. *Let* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ *be a non-interactive commitment scheme. Let* Π_{cWE} *be a witness encryption over commitments for some circuit class* $\mathcal{C}$ *over commitment scheme* C*. If* Π_{cWE} *has weak semantic security, and* C *has hiding property, then* Π_{cWE} *has strong semantic security.*

Proof. Assume $\mathcal{A}$ is a PPT adversary against strong semantic security. We construct an efficient adversary $\mathcal{B}$ that breaks weak semantic security of Π_{cWE} with non-negligible advantage.

First, $\mathcal{B}$ runs $\mathcal{A}$ with the commitment key ck received from the challenger. $\mathcal{B}$ then simulates the oracle $\mathcal{O}_{\mathsf{com}}$ for $\mathcal{A}$ in the natural way. Namely, for any input s_i, it outputs $\mathsf{cm}_i \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}_i; \rho_i)$ for some randomness $\rho_i \in \mathcal{S}_r$, and stores cm_i in $\mathcal{Q}$. Upon receiving a tuple $(\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1)$ from $\mathcal{A}$, $\mathcal{B}$ forwards the tuple to the challenger. Upon receiving the challenge ciphertext ct (for the encryption of m_b) from the challenger, $\mathcal{B}$ generates a ciphertext ct_i for each commitment $\mathsf{cm}_i \in \mathcal{Q}$. To do so, $\mathcal{B}$ selects $c \leftarrow\$ \{0,1\}$ and computes $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}_i, C, y), m_c)$ for any $\mathsf{cm}_i \in \mathcal{Q}$. Next, $\mathcal{B}$ checks whether $C(\mathsf{s}) \neq y$, and if so forwards $\mathbf{ct} := \{\mathsf{ct}\} \cup \{\mathsf{ct}_i\}_{i \in [|\mathcal{Q}|]}$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ outputs a random guess $b' \leftarrow\$ \{0,1\}$ for the bit b. Finally, upon receiving a guess b' from $\mathcal{A}$, $\mathcal{B}$ forwards b' to the challenger. It is easy to see that if $c = b$, then $\mathcal{B}$ is perfectly simulating strong semantic security game for $\mathcal{A}$.

To prove the lemma, we define $|\mathcal{Q}| + 2$ hybrid distributions such that the first hybrid corresponds to the strong semantic security and the last hybrid corresponds to the above

game simulated by $\mathcal{B}$. We conclude the proof by showing that an adversary with non-negligible advantage in the first hybrid implies the existence of an efficient adversary with non-negligible advantage in the last hybrid.

Hybrid 0. This is the strong semantic security game. Namely,

1. The adversary $\mathcal{A}$ receives the commitment key ck, where $\mathsf{ck} \leftarrow \mathsf{C.Setup}(1^\lambda)$.

2. $\mathcal{A}$ adaptively makes commitment queries for messages s_i, and for each receives $\mathsf{cm}_i \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}_i; \rho_i)$.

3. After some number of queries listed in $\mathcal{Q}$, $\mathcal{A}$ outputs a tuple $(\mathsf{st}, \mathsf{s}, \rho, C, y, m_0, m_1)$ for which $C(\mathsf{s}) \neq y$. The challenger samples a random bit $b \leftarrow\!\!\$\, \{0,1\}$, generates encryptions of m_b via $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, y), m_b)$, and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}_i, C, y), m_b)$ for all $\mathsf{cm}_i \in \mathcal{Q}$, and sends $\mathbf{ct} := \{\mathsf{ct}\} \cup \{\mathsf{ct}_i\}_{i \in [|\mathcal{Q}|]}$ to $\mathcal{A}$ as the challenge ciphertext.

4. Eventually, $\mathcal{A}$ outputs a guess b' for the bit b.

Hybrids $\mathsf{k} = 1, \ldots, |\mathcal{Q}|$. Same as the previous hybrid, except the first k ciphertexts $\{\mathsf{ct}_i\}_{i \in [k]}$ are computed with respect to cm_i being a commitment of s. Namely,

1. *Identical to Hybrid 0.*

2. *Identical to Hybrid 0.*

3. The challenger samples a random bit $b \leftarrow\!\!\$\, \{0,1\}$, generates encryptions of m_b via $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, y), m_b)$, and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}_i, C, y), m_b)$ $(i = 1, \ldots, |\mathcal{Q}|)$ computed as before, except in the first k ciphertexts $\{\mathsf{ct}_i\}_{i \in [k]}$, the commitment cm_i is computed as $\mathsf{cm}_i \leftarrow \mathsf{C.Commit}(\mathsf{ck}, \mathsf{s}; \rho_i)$ for some randomness $\rho_i \in \mathcal{S}_r$.

4. *Identical to Hybrid 0.*

Hybrid $\mathsf{k} = |\mathcal{Q}| + 1$. Same as the previous hybrid, except the ciphertexts $\{\mathsf{ct}_i\}_{i \in [|\mathcal{Q}|]}$ are encryptions of m_c for a uniformly random $c \leftarrow\!\!\$\, \{0,1\}$. Namely,

1. *Identical to Hybrid $|\mathcal{Q}|$.*

2. *Identical to Hybrid $|\mathcal{Q}|$.*

3. The challenger samples random bits $b \leftarrow\!\!\$\, \{0,1\}$ and $c \leftarrow\!\!\$\, \{0,1\}$, and generates encryptions of m_b and m_c respectively via $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}, C, y), m_b)$, and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{ck}, (\mathsf{cm}_i, C, y), m_c)$ $(i = 1, \ldots, |\mathcal{Q}|)$.

4. *Identical to Hybrid $|\mathcal{Q}|$.*

For $k = 0, \ldots, |\mathcal{Q}| + 1$, denote by adv_i the advantage of $\mathcal{A}$ in guessing the bit b in Hybrid i. It is easy to see that for any $0 \leq i < |\mathcal{Q}|$, we have $|\mathsf{adv}_i - \mathsf{adv}_{i+1}| \leq \mu(\lambda)(\lambda)$, where $\mu(\lambda)(\lambda)$ is some negligible function. This is because the distributions $\mathcal{D}_i$ and $\mathcal{D}_{i+1}$ respectively defined by the hybrids i and $i+1$ only differ in their $(i+1)$-th ciphertext,

which is the encryption of m_b under a commitment of s_{i+1} for $\mathcal{D}_i$ and encryption of m_b under a commitment of s for $\mathcal{D}_{i+1}$. Thus, the difference $|\mathsf{adv}_i - \mathsf{adv}_{i+1}|$ is exactly equal to the adversary's advantage in the hiding experiment. By the hiding property of the commitment scheme, it thus follows that this difference is negligible.
Furthermore, observe that in the last hybrid, $c = b$ with probability $1/2$ and hence we have that with probability at least $1/2$, the two distributions $\mathcal{D}_{|\mathcal{Q}|}$ and $\mathcal{D}_{|\mathcal{Q}|+1}$ are identical. This, together with the fact that $\mathcal{D}_0$ and $\mathcal{D}_{|\mathcal{Q}|}$ have a negligible difference imply that having an efficient adversary with non-negligible advantage ε against hybrid 0 results in a non-negligible advantage $\varepsilon/2$ against hybrid $|\mathcal{Q}| + 1$. This completes the proof of the lemma. □

6.11.2 Proof of Theorem 6.1

Theorem 6.4 (Theorem 6.1 (restated))**.** *Let* $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit})$ *be a non-interactive extractable commitment scheme and* $\Pi_{\mathsf{cWE}} = (\mathsf{Enc}_{\mathsf{cWE}}, \mathsf{Dec}_{\mathsf{cWE}})$ *be a strong semantically secure witness encryption scheme over* C *for a circuit class* $\mathcal{C}$ *encoding the lottery predicate* $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ *as defined in Section 6.3. Let* Γ *be a blockchain protocol as defined in Section 4.2.* Π_{ECW} *is an* IND-CPA*-secure ECW scheme as per Definition 6.2.*

Proof. Assume by contradiction that there exists an adversary $\mathcal{A}_{\mathsf{ECW}}$ with non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ in the ECW setting as described in Section 6.2.1. We construct an adversary $\mathcal{A}_{\mathsf{cWE}}$ with black-box access to $\mathcal{A}_{\mathsf{ECW}}$ that has non-negligible advantage in breaking strong semantic security of Π_{cWE} as defined above. We assume (w.l.o.g.) that $\mathcal{A}_{\mathsf{ECW}}$ only corrupts one party P_a[5]. $\mathcal{A}_{\mathsf{cWE}}$ proceeds as follows:

1. Upon receiving the commitment key ck from the challenge, $\mathcal{A}_{\mathsf{cWE}}$ proceeds as follows:

   a) $\mathcal{A}_{\mathsf{cWE}}$ acts as the environment $\mathcal{Z}$ orchestrating the execution of the blockchain protocol Γ towards $\mathcal{A}_{\mathsf{ECW}}$, placing the commitment key ck in the genesis block. $\mathcal{A}_{\mathsf{cWE}}$ acts exactly as $\mathcal{Z}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$.

   b) $\mathcal{A}_{\mathsf{cWE}}$ simulates honest parties P_h executing the setup phase and publishing a valid $(\mathsf{cm}_h, \sigma_h)$ on the blockchain. To simulate cm_h for each honest party, $\mathcal{A}_{\mathsf{cWE}}$ calls the oracle $\mathcal{O}_{\mathsf{com}}$ on some random input $\mathsf{sk}_{L,h}$ and sets cm_h to be $\mathcal{O}_{\mathsf{com}}$'s output.

   c) At some point, $\mathcal{A}_{\mathsf{ECW}}$ outputs challenge parameters $\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_0, m_1$ from its view of the blockchain. $\mathcal{A}_{\mathsf{cWE}}$ constructs a circuit C that encodes the predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$, where $\mathbf{B}$, sl and R are hardcoded and $\mathsf{sk}_{L,i}$ is the witness.

   d) Finally, if there exists a valid setup message $(\mathsf{cm}_a, \sigma_a)$ published in the common prefix $\mathbf{B}^{\lceil\kappa}$ by P_a (*i.e.* the corrupted party P_a is in $\mathcal{P}_{Setup}$), $\mathcal{A}_{\mathsf{cWE}}$ extracts $\mathsf{sk}_{L,a}, \rho_a$ from cm_a using the extractability of the commitment scheme C and

[5] In reality there will be more than one corrupted party; the main argument underlying our proof holds regardless.

outputs $(\mathsf{st}, \mathsf{sk}_{L,a}, \rho_a, C, 1, m_0, m_1)$ to the challenger. Otherwise, $\mathcal{A}_{\mathsf{cWE}}$ outputs $(\mathsf{st}, \mathsf{sk}_{L,k}, \rho_k, C, 1, m_0, m_1)$ to the challenger, where $\mathsf{sk}_{L,k}, \rho_k$ are chosen at random and such that $C(\mathsf{sk}_{L,k}) \neq 1$.

2. Upon receiving ciphertexts $\mathbf{ct} = \{\mathsf{ct}\} \cup \{\mathsf{ct}_h\}_{P_h \in \mathcal{P}_{Setup}}$ from the challenger, if $P_a \in \mathcal{P}_{Setup}$, then $\mathsf{ct} = \mathsf{ct}_a$ was computed w.r.t. P_a's commitment cm_a and ct_h computed w.r.t. the honest party's commitment cm_h. Otherwise, if only honest parties are in $\mathcal{P}_{Setup}$, $\mathcal{A}_{\mathsf{cWE}}$ forwards the ECW ciphertexts $\mathbf{ct} = \{\mathsf{ct}_h\}_{P_h \in \mathcal{P}_{Setup}}$ to $\mathcal{A}_{\mathsf{ECW}}$. $\mathcal{A}_{\mathsf{cWE}}$ continues the execution of Γ with $\mathcal{A}_{\mathsf{ECW}}$ from the round where it stopped when $\mathcal{A}_{\mathsf{ECW}}$ outputted challenge parameters $\mathbf{B}, \mathsf{sl}, \mathsf{R}, m_0, m_1$.

3. Upon receiving a guess b' from $\mathcal{A}_{\mathsf{ECW}}$, $\mathcal{A}_{\mathsf{cWE}}$ forwards b' to the challenger.

First, notice that $\mathcal{A}_{\mathsf{ECW}}$ has the same access to the underlying blockchain protocol Γ (and to the system parameters in the genesis block) as in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$. In case $\mathcal{A}_{\mathsf{ECW}}$ provided a valid setup message, it receives $\mathbf{ct}$ containing a cWE ciphertext ct_a generated with respect to its commitment cm_a and the circuit encoding the lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$, where $\mathbf{B}$, sl and R provided by $\mathcal{A}_{\mathsf{ECW}}$ are hardwired. Moreover, $\mathbf{ct}$ contains ciphertexts ct_h for each cm_h, encrypting the same m_b as in ct_a. Hence, $\mathbf{ct}$ is distributed exactly as in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$. If $\mathcal{A}_{\mathsf{ECW}}$ has non-negligible advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$, it is able to distinguish whether ct_a contains m_0 or m_1 with non-negligible advantage even though it does not have $\mathsf{sk}_{L,a}$ and $\mathsf{cm}_a \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathsf{sk}_{L,a}; \rho_a)$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,a}) = 1$, *i.e.* it does not have $\mathsf{sk}_{L,a}$ such that $C(\mathsf{sk}_{L,a}) = 1$. This means that, by forwarding guess b' from $\mathcal{A}_{\mathsf{ECW}}$, $\mathcal{A}_{\mathsf{cWE}}$ in the cWE semantic security game has the same advantage as $\mathcal{A}_{\mathsf{ECW}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$. In case it did not provide a valid setup message, $\mathcal{A}_{\mathsf{ECW}}$ only sees $\mathbf{ct} = \{\mathsf{ct}_h\}_{P_h \in \mathcal{P}_{Setup}}$ with ct_h being an encryption of m_b with respect to the commitments cm_h for which it does not know the opening. Hence, $\mathbf{ct}$ is again distributed exactly as in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ with probability 1. In this case, by an analogous argument as before, the advantage of the adversary $\mathcal{A}_{\mathsf{cWE}}$ must be the same as the advantage of the adversary $\mathcal{A}_{\mathsf{ECW}}$ in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$.

Since we assume that $\mathcal{A}_{\mathsf{ECW}}$ has a non-negligible advantage, $\mathcal{A}_{\mathsf{cWE}}$ will also obtain a non-negligible advantage and thus break the cWE scheme we assume is secure. Hence, Π_{ECW} is an IND-CPA-secure ECW scheme. □

6.12 Further Details on YOSO MPC

6.12.1 More Efficient AfP based on VRF

6.12.1.1 VRF-based Lottery.

This section introduces a specific lottery mechanism which will be the underlying lottery predicate for the AfP described in the next section. The backbone of the lottery is a VRF scheme VRF as described in [DGKR18]. This VRF has the properties of simulatability and

unpredictability under malicious key generation which will become useful when arguing about security of the AfP. The VRF scheme is a tuple $(\mathsf{VRF.Gen}, \mathsf{VRF.Prove}, \mathsf{VRF.Verify})$ where $\mathsf{VRF.Gen}(1^\kappa)$ outputs a pair of keys $(\mathsf{VRF.pk}, \mathsf{VRF.sk})$. The VRF.Prove takes as input a value x and outputs a pair $(y, \pi) \leftarrow \mathsf{VRF.Prove}_{\mathsf{VRF.sk}}(x)$ which is the output value y and the correctness certificate π. The verification is then done by evaluating $\mathsf{VRF.Verify}_{\mathsf{VRF.pk}}(x, y, \pi)$ which outputs 1 iff π attests to the correctness of y as the output of the VRF evaluated on x with key VRF.sk.
We recall the blockchain setup described in Section 4.4.2 where each party P_i is represented by a pair $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$ associated with public data $(\mathsf{Sig.pk}_i, \mathsf{aux}_i, \mathsf{stake}_i)$. Let aux_i contain a VRF public key $\mathsf{VRF.pk}_i$ as described above and let the lottery secret key be $\mathsf{sk}_{L,i} = (\mathsf{Sig.pk}_i, \mathsf{VRF.sk}_i)$.
Finally, we introduce a function $\mathsf{param}(\mathbf{B}, \mathsf{sl})$. This function outputs a tuple $(\{\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i\}_{i\in[n]}, \eta, \phi)$ associated with the specific blockchain $\mathbf{B}$ and slot sl. Beyond obtaining the public information $(\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i)$ the function also returns a nonce, η, as well as a public function $\phi(\cdot)$ which on input stake_i computes the threshold for winning the lottery.
The lottery predicate based on the VRF is described in Algorithm 14.

Algorithm 14: $\mathsf{lottery}_{\mathsf{VRF}}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,j})$

$(\{\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i\}_{i\in[n]}, \eta, \phi) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$
$(\mathsf{Sig.pk}_j, \mathsf{VRF.sk}_j) \leftarrow \mathsf{sk}_{L,j}$
$(y, \pi) \leftarrow \mathsf{VRF.Prove}_{\mathsf{sk}_j}(\mathsf{sl}||\mathsf{R}||\eta)$
if $y < \phi(\mathsf{stake}_j)$ **then**
    if $\mathsf{VRF.Verify}_{\mathsf{pk}_j}(\mathsf{sl}||\mathsf{R}||\eta, y, \pi) =$ **then**
        return 1
    end if
end if
return 0

6.12.1.2 VRF-based AfP

With the VRF-based lottery $\mathsf{lottery}_{\mathsf{VRF}}$ in place, we are now ready to introduce the VRF-based AfP. We first note that our general approach of applying a SoK for the knowledge of a secret key still applies. However, using the structure of the lottery, and in particular the VRF, allows for a much more efficient AfP which has applications in most PoS settings as well.
The AfP scheme uses a NIZKPoK which has a setup executed as a part of the blockchain setup such that the CRS is in the genesis block. The algorithms for the scheme are $\pi \leftarrow \mathsf{NIZKPoK.Prove}(\mathsf{crs}, \mathtt{x}, \mathtt{w})$ and $\{0, 1\} \leftarrow \mathsf{NIZKPoK.Verify}(\mathsf{crs}, \mathtt{x}, \pi)$.

The VRF-based AfP protocol Π_{AfP} is described below.

Authenticate. $\sigma \leftarrow \Pi_{\mathsf{AfP}}.\mathsf{Sign}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_{L,j}, m)$ To authenticate a message, m, a party first checks that $\mathsf{lottery}_{\mathsf{VRF}}(\mathbf{B}, \mathsf{sl}, \mathsf{S}, \mathsf{sk}_{L,j}) = 1$. It then obtains the output and cer-

tificate $(y, \pi_{\mathsf{VRF}}) \leftarrow \mathsf{VRF.Prove}_{\mathsf{VRF.sk}_j}(\mathsf{sl}||\mathsf{R}||\eta)$. Finally, it produces $\pi_{\mathsf{NIZKPoK}} \leftarrow \mathsf{NIZKPoK.Prove}\{\sigma_{\mathsf{Sig}} \mid \mathsf{Sig.Verify}_{\mathsf{Sig.pk}_j}(\sigma_{\mathsf{Sig}}, m) = 1\}$ which is a NIZK-PoK of a signature produced under $\mathsf{Sig.sk}_j$.
It then outputs a tuple $\sigma_{\mathsf{AfP}} \leftarrow (\mathsf{Sig.pk}_j, y, \pi_{\mathsf{VRF}}, \pi_{\mathsf{NIZKPoK}})$

Verify. $\{0,1\} \leftarrow \Pi_{\mathsf{AfP}}.\mathsf{Verify}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{S}, \sigma, m)$ To verify an AfP tag the verifier obtains parameters from the blockchain $(\{\mathsf{Sig.pk}_i, \mathsf{VRF.pk}_i, \mathsf{stake}_i\}_{i\in[n]}, \eta, \phi) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$. It then parses the tag as $\sigma_{\mathsf{AfP}} \leftarrow (\mathsf{Sig.pk}_j, y, \pi_{\mathsf{VRF}}, \pi_{\mathsf{NIZKPoK}})$ and gets the VRF verification key $\mathsf{VRF.pk}_j$ for the party that the AfP points to. It then checks the following

1. Makes sure that $\mathsf{VRF.Verify}_{\mathsf{VRF.pk}_j}(\mathsf{sl}||\mathsf{R}||\eta, y, \pi_{\mathsf{VRF}}) = 1$ *i.e.* the VRF output was correctly generated under lottery key of party P_j.

2. Checks that $\mathsf{NIZKPoK.Verify}(\pi_{\mathsf{NIZKPoK}}, (\mathsf{Sig.pk}_j, m)) = 1$ which verifies the proof of signature knowledge.

3. And $y < \phi(\mathsf{stake}_j)$ which makes sure that the lottery was conducted correctly with the stake of P_j.

If all checks go through, the algorithm outputs 1. Otherwise, it outputs 0.

Theorem 6.5. *Let* VRF *be the VRF scheme described in [DGKR18] with a secure NIZK-PoK scheme* NIZKPoK*. The protocol* Π_{AfP} *(described above) running in the context of a blockchain protocol* Γ *with underlying lottery* $\mathsf{lottery}_{\mathsf{VRF}}$ *(Algorithm 14) is an AfP scheme according to Definition 6.5.*

Proof. (Sketch) Assume that an adversary $\mathcal{A}$ obtains a non-negligible advantage in $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U}}$. In other words, $\mathcal{A}$ is able to forge an AfP tag with noticeable probability. We claim that such an adversary can do at least one of three things:

1. It can forge a signature under $\mathsf{Sig.sk}_i$, thus violating the EUf-CMA security of the signature scheme.

2. It can produce a convincing proof π_{NIZKPoK} of knowledge of a signature produced with a signature secret key where the corresponding lottery secret key did not win the lottery. Since we assume that only P_i knows the pair $(\mathsf{Sig.sk}_i, \mathsf{sk}_{L,i})$, such a convincing proof must violate the soundness of the NIZKPoK scheme.

3. It can forge a VRF certificate such that the VRF.Verify algorithm accepts a certificate π_{VRF} under a different $y' \neq y$ when evaluated with the VRF public key VRF.pk of the adversary and thus convinces the authenticator. This violates the simulator of the simulatable VRF introduced in [DGKR18].

Since we assume that NIZKPoK is a secure NIZK-PoK scheme and VRF a secure scheme based on the functionality in [DGKR18], we conclude that Π_{AfP} is secure with respect to Definition 6.5. □

6.12.1.3 AfP Privacy.

This simple AfP mechanism for a VRF-based lottery predicate does not only satisfy the existential unforgeability definition of an AfP. It has AfP privacy.

Theorem 6.6. *The AfP protocol* Π_{AfP} *described above with underlying lottery predicate* $\mathsf{lottery}_{\mathsf{VRF}}$ *running in the context of blockchain* Γ *has AfP privacy.*

Proof (Sketch).
We use the notation $\mathcal{D}_0$ and $\mathcal{D}_1$ for the distribution of outputs when the adversary is put in scenario 0 and scenario 1, respectively. Our aim is to show the existence of a simulator such that $\mathcal{D}_0$ and $\mathcal{D}_1$ are computational indistinguishable.
We introduce 4 hybrids $\{\mathsf{H}_i\}_{i=1,\dots,4}$ where $\mathsf{H}_1 = \mathcal{D}_0$ and $\mathsf{H}_4 = \mathcal{D}_1$.

- H_2 This hybrid is identical to H_1 but we use the simulator of the NIZKPoK scheme to simulate the proof of signature knowledge that convinces. Due to the security of the NIZKPoK scheme H_2 and H_1 are indistinguishable to the PPT adversary $\mathcal{A}$.

- H_3 The difference from H_2 is that instead of invoking the VRF scheme VRF we are using the simulatability of the construction to output valid proofs.

- H_4 This hybrid does not need access to any lottery winning secret keys and thus can be completely simulated by Sim. It is still necessary to observe the distribution of the stake to correctly simulate the output of the oracle $\mathcal{O}_{\mathsf{AfP}}$.

Assume that an adversary can distinguish $\mathcal{D}_0$ and $\mathcal{D}_1$ with non-negligible probability ρ. It implies that there exists an $i \in \{1, 2, 3\}$ such that H_i and H_{i+1} can be distinguished with non-negligible probability at least $\rho/3$. This contradicts the indistinguishability of hybrids. Thus, we conclude that the distributions $\mathcal{D}_0$ and $\mathcal{D}_1$ are computationally indistinguishable due to the simulator Sim obtained through the sequence of hybrids above.

6.12.2 Extended Lotteries

So far the probability of winning a role can depend on the slot sl, the role R, and the lottery witness $\mathsf{sk}_{L,i}$. The dependence can be arbitrarily complex. We sometimes need to assume that the lottery shows some level of structure to be able to ensure that a given set of roles has enough honest machines winning them.

6.12.2.1 Smooth Lotteries.

First we define that a lottery has individual winning probabilities if for a given sl and a given account sk there exist a probability p such that it holds for all R that $\Pr[\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1] \approx p$. This is the case for most PoS lotteries as the probability of winning depends only on the stake that $\mathsf{sk}_{L,i}$ has in a given slot.

We will also need to assume independence of winning events. It is useless for the sake of using, e.g., the law of large numbers if in a given slot either all honest parties win a role or none win a role. We typically needed that except with negligible probability some large enough fraction wins roles.

We require that for all sl and all $(\mathsf{R}_i, \mathsf{sk}_{L,i})$ and $(\mathsf{R}_j, \mathsf{sk}_{L,j}) \neq (\mathsf{R}, \mathsf{sk}_{L,i}$ it holds that $\Pr[\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,i}) = 1 \mid \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_j, \mathsf{sk}_{L,j})] \approx \Pr[\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,i}) = 1]$. We extend this to n-independence where the probability of an account winning a role does not depend on the outcome of $n-1$ other lotteries in the same slot.

We call a lottery with individual winning probabilities and n-independence an n-smooth lottery. For an n-smooth lottery we can compute the probability that a set of up to n roles are won by honest parties directly from the individual winning probabilities of the slot.

6.12.2.2 Hardness Adjustment.

We will sometimes need to assume that the hardness of the lottery can be adjusted. This can in principle be captured in the current formalism $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ as we could have some roles be harder to win. This would however ruin individual winning probabilities so we prefer an explicit notation for it. We assume a new parameter $\mathsf{hard} \in [0, 1]$ which can be used to control hardness of the lottery. For simplicity assume that $\mathsf{hard} \in [0, 1]$. We require that

$$\Pr[\exists \mathsf{sk}\,(\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}, \mathsf{hard}) = 1)] \approx \mathsf{hard}\ .$$

A more realistic model would have to assume that the probability can be controlled to be in some interval, e.g., $[\mathsf{hard}/2, 2\mathsf{hard}]$, but nothing essential is lost in assuming the simplistic model in this work where the focus is on EtF and not intricacies of the lotteries themselves. Scaling of hardness is typically easy to construct as most PoS lotteries give each party a pseudo-random number and say that the party won if the number is below some threshold. One can use hard to adjust the threshold. We assume that adjusting the hardness maintains n-smoothness.

6.12.2.3 Filtering.

Another possible extension is having an extra parameter filter which is a PPT predicate filtering the lottery. Given an account $\mathsf{sk}_{L,i}$ we assume it can be computed in PPT from the information on $\mathbf{B}$ and the public parameters $\mathsf{pk}_{L,i}$ associated to $\mathsf{sk}_{L,i}$. In particular, filter does not need $\mathsf{sk}_{L,i}$ to be efficiently computable. We require that $\mathsf{filter}(\mathbf{B}, \mathsf{sk}_{L,i})$ outputs $\top$ or $\bot$. We require from the lottery that if $\mathsf{filter}(\mathbf{B}, \mathsf{sk}_{L,i}) = \bot$ then $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}, \mathsf{filter}) = 0$. If $\mathsf{filter}(\mathbf{B}, \mathsf{sk}_{L,i}) = \top$, then we require that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}, \mathsf{filter}) = \mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$. Since the filter can be computed in PPT given the blockchain it is typically trivial to augment existing lotteries with a filter. We can simply let the lottery predicate include a check of the filter. We could again capture this in the existing formalism simply by letting $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ ignore the filtered winners, but this would again ruin individual winning probabilities of the underlying mechanism.

6.12.2.4 Honest Majority.

When realizing YOSO MPC, we must show that a lottery selects parties for roles in such a way that achieve honest majority among roles, according to the corruption statuses defined in Section 6.5.4. For simplicity of the discussion we assume an n-smooth lottery, where we can control the hardness. Assume that we set the hardness such that a given role is won with probability ϕ. It is easy to see that when we have n-smoothness then if there is probability ϕ that a role is won, then there is about probability ϕ^2 that it is won twice, giving a MALICIOUS role. Clearly there is probability $1-\phi$ that it is not won, giving a CRASHED role. The expression $\phi^2+1-\phi$ has a minimum of 75%, so 75% of all roles will be malicious or crashed even if we start with perfect honesty. We can therefore never expect to get honest majority. The trick is to design protocols which can tolerate many crashed parties as long as there are more honest parties than corrupted parties among the non-crashed parties.

Assume a lottery where a unique winner is honest with probability $\frac{1}{2}+\epsilon$. In that case the probability that a role is won by a single honest winner is $\phi(\frac{1}{2}+\epsilon)$. The probability that the role is won more than once is about $\phi/(1-\phi)$ by an application of a geometric series. The probability that it is won by a single corrupted party is $\phi(\frac{1}{2}-\epsilon)$. To have more honest parties than corrupted parties in expectation we therefore need that $\phi(\frac{1}{2}+\epsilon) > \phi/(1-\phi)+\phi(\frac{1}{2}-\epsilon)$, which solves to $\phi > 1-2\epsilon$. By picking $\phi = 1-2\epsilon+\delta$ for a positive constant we get that the expected number of honest parties is $h = \phi(\frac{1}{2}+\epsilon)n$ and that the expected number of corrupted parties is $t = (\phi/(1-\phi)+\phi(\frac{1}{2}-\epsilon))n$ and that $h-t > 2\gamma n$ for a positive constant γ.

By setting $H = h-\gamma n$ and using a Chernoff bound and n-smoothness we can pick n large enough to ensure that there are more than H honest parties except with negligible probability. By setting $T = t+\gamma n$ and picking n large enough we can similarly ensure that there are less that T corrupted parties except with negligible probability. Note that $H > T$. Hence, we can satisfy Definition 6.7 of Section 6.5.4.

Chapter 7

YOLO YOSO [CDGK22]

7.1 Overview

In this section we highlight the main technical components of our contributions. We remark that our main goal is providing simple constructions that yield efficient instantiations of PVSS towards anonymous committees along with efficient AfP schemes allowing parties to prove they received shares sent to a given role.

Encryption to the Future We introduce a simple ECW protocol where each party chooses a key pair in the system and then a mixnet is used to anonymize them. We can then define a simple lottery predicate that selects one of these keys. The winner of the lottery can trivially know that they have won this lottery. By combining this with an IND-CPA encryption scheme that encrypts a message under that key, we can obtain IND-CPA ECW. Using a homomorphic encryption scheme we can also encrypt to multiple lottery winners and prove that the same message is received by all of them.

Authentication from the Past

The Easy Way: An easy way of obtaining reusable ECW setup is to repeat the lottery setup and obtain multiple anonymized keys for each party. Then, any party can use a new anonymized public key for each AFP tag. This ensures that the AFP scheme can be executed a bounded number times before lottery winners can be linked to specific public keys in the setup and ciphertexts starts betraying their receivers.

The Reusable Way: In Section 7.9, we show that a party can prove membership in a given committee without needing to reveal its role in this committee. This is done by signing a message with a ring signature [RST01] where the secret key corresponds to a public key in the committee. These signatures hide the identity of the party. Moreover, we require the signature to be linkable [LWW04], so that no two parties can claim the same secret key. Using this and an anonymous channel, we can construct an AfP that can be used multiple times without linking a party P_i to its setup public key. More interestingly, we also present a protocol that leverages the presence of a dealer (which could be a party that encrypted the message to that committee) to reduce the size of these proofs

of membership to constant (for the parties making the claims). This uses Camenisch-Lysyanskaya signatures [CL04], where the dealer signs the public keys of the committee, and the parties can then "complete" one of these signatures without revealing which one. We introduce a simple linkable version of these signatures.

PVSS We introduce two constructions for PVSS. The first, HEPVSS, is based on a generic encryption scheme which enjoys certain linearity properties with respect to encryption and decryption, and has the advantage that the security of the PVSS can be based on IND-CPA security of the scheme. The homomorphic properties of the scheme allow for simple proofs of sharing correctness and reconstruction. While we are only aware of El Gamal scheme satisfying the notion of the homomorphic properties we need, we hope that a relaxed version of this abstraction allows to capture other encryption schemes with homomorphic properties such as latticed-based assumptions or Paillier in future work. In our second scheme DHPVSS, we introduce the idea of providing the dealer with an additional key pair for share distribution. This idea is powerful in combination with a technique used in SCRAPE to prove that encrypted shares lie on a polynomial of the right degree. The novelty is that, while in SCRAPE this needed an additional discrete logarithm equality (DLEQ) proof *for each share*, our new scheme requires *a single DLEQ proof*. This reduces the sharing correctness proof to only 2 $\mathbb{Z}_p$-elements while each encrypted shares is still one group element.

We also introduce PVSS resharing protocols for both constructions, where a committee, among which a secret is PVSSed, can create shares of the same secret for the next committee, in a publicly verifiable way.

PVSS Towards Anonymous Committees Finally, we show that we can replace standard encryption and authentication in our PVSS protocols by ECW and AFP and thereby obtain PVSS toward anonymous committees.

7.2 ECW based on $\mathbb{Z}_p$-Linearly Homomorphic Encryption

This section presents an ECW protocol based on a $\mathbb{Z}_p$-linearly homomorphic encryption scheme described in Section 5.8.3 and a mixnet (Section 5.10). Together with the ECW, we introduce an AfP scheme - a mechanism that allows a committee member to authenticate messages. The two schemes will be the backbone of the anonymous PVSS presented in Section 7.5. Before presenting the actual ECW and AfP protocols, we introduce the underlying lottery predicate that will be the cornerstone in our two schemes.

7.2.1 Lottery Predicate

We assume a running blockchain as described Section 4.2 and a function param that has access to the blockchain state. During the setup, each party samples an encryption key

pair $(\mathsf{sk}_{\mathcal{E},i}, \mathsf{pk}_{\mathcal{E},i})$ and inputs $\mathsf{pk}_{\mathcal{E},i}$ to the mixnet (Section 5.10). The output of the mixnet is a tuple $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party through param function. The function param takes as input the blockchain $\mathbf{B}$ and the slot sl and outputs a tuple $(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$. Here, $(j, \mathsf{pk}_{\mathsf{Anon},j})$ is equal to $(\psi(i), \mathsf{pk}_{\mathcal{E},i})$ for the permutation ψ defined by the mixnet. Finally, η is the public randomness from the blockchain corresponding to $\mathbf{B}$ and sl. Note, that only the owner of $\mathsf{sk}_{\mathcal{E},i}$ knows j such that $\mathsf{pk}_{\mathsf{Anon},j} = \mathsf{pk}_{\mathcal{E},i}$. Let $\mathcal{H} : \{0,1\}^* \to [n]$ be a hash function that outputs a number that points to a specific index in the list of public keys. The lottery predicate $\mathsf{lottery}$ is detailed below.

Algorithm 15: $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$

1: $(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl})$
2: $(\mathsf{pk}_{\mathcal{E},i}, \mathsf{sk}_{\mathcal{E},i}) \leftarrow \mathsf{sk}_{L,i}$
3: $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{R}||\eta)$
4: **return** 1 iff $\mathsf{pk}_{\mathcal{E},i} = \mathsf{pk}_{\mathsf{Anon},k}$

It is easy to see that the lottery described above associates a *single* party (from the set of eligible parties) with the role R. Furthermore, the party can locally check if it won the lottery by checking that the output of the hash function points to its own public key in the permuted set. Crucially, the party winning the lottery can stay covert since no other party can link the winning lottery key to the owner of the corresponding secret key. These properties will be useful when we want to encrypt shares towards an anonymous committee.

7.2.2 ECW Protocol

This section introduces a ECW protocol (Fig. 7.1) based on the lottery predicate presented in Section 7.2.1. We note that ECW is just a restricted version of EtF where the lottery is conducted wrt. the *current* blockchain $\mathbf{B}$ and slot sl. Thus, all definitions in Section 6.1 applies to ECW schemes too.

ECW Protocol

Public parameters: A prime p, a $\mathbb{Z}_p$-linearly homomorphic encryption scheme $\mathcal{E} = (\mathcal{E}.g, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ with notation as in Section 5.8.3 and a lottery as described in Section 7.2.1.

Set-up:

1. Every party runs $\mathcal{E}.g()$ obtaining a key pair $(\mathsf{sk}_{\mathcal{E},i}, \mathsf{pk}_{\mathcal{E},i})$.
2. Each party inputs $\mathsf{pk}_{\mathcal{E},i}$ to the mixnet. The output of the mixnet is a tuple $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party when using the param function.

Encryption protocol: Input $(\mathbf{B}, \mathsf{sl}, \mathsf{R})$ and $m \in \mathfrak{P}$.

1. Run $\mathsf{param}(\mathbf{B}, \mathsf{sl})$ and obtain $(\{(I, \mathsf{pk}_{\mathsf{Anon},I})\}_{I\in[n]}, \eta)$.
2. Obtain random index by $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{R}||\eta)$.
3. Choose ρ in $\mathfrak{R}$ and set $c = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},k}}(m, \rho)$.
4. Sender outputs c.

Decryption protocol: Input for party i is $\mathbf{B}, \mathsf{sk}_{L,i}$ and c.

1. Checks that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$.
2. Outputs $m = \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}_{\mathsf{Anon},i}}(c)$.

Figure 7.1: ECW Protocol

Theorem 7.1 (IND-CPA ECW). *Let $\mathcal{E}$ be an IND-CPA secure $\mathbb{Z}_p$-linearly homomorphic encryption scheme. The construction in Fig. 7.1 with lottery predicate as in Section 7.2.1 is an IND-CPA secure ECW (as in Definition 6.2).*

(See proof sketch in Section 7.6)

7.2.3 AfP Protocol

In this section we present our protocol for AfP (Definition 6.5). It is described in Fig. 7.2 and is based on a Signature of Knowledge (SoK) [CL06]. A SoK scheme is a pair of algorithms $(\mathsf{SoK.sign}, \mathsf{SoK.verify})$ and is defined in context of a relation R. We consider statements of the form $x = (\mathbf{B}, \mathsf{sl}, \mathsf{R})$ and witnesses $w = \mathsf{sk}$. We say that $R(x = (\mathbf{B}, \mathsf{sl}, \mathsf{R}), w = \mathsf{sk}) = 1$ iff $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}) = 1$. A signature is produced by running $\sigma \leftarrow \mathsf{SoK.sign}(x, w, m)$. And it can be verified by checking that the output of $\mathsf{SoK.verify}(x, \sigma, m)$ is 1. Our AfP uses the SoK to sign m under the knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$. This will exactly attest that the message m was sent by the winner of the lottery for R. An instantiation of this AfP protocol could use DL proofs (Section 5.7).

AfP Protocol

Public parameters and **Set-up** as described in Fig. 7.1 plus additional setup for the SoK scheme $\mathsf{SoK} = (\mathsf{SoK.sign}, \mathsf{SoK.verify})$.
Authentication protocol: Input for party i is $(\mathbf{B}, \mathsf{sl}, \mathsf{R})$ and $m \in \mathfrak{P}$.

1. Checks that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$.
2. Constructs an SoK on the message m of knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$ resulting in $\sigma_{\mathsf{SoK}} \leftarrow \mathsf{SoK.sign}((\mathbf{B}, \mathsf{sl}, \mathsf{R}), \mathsf{sk}_{L,i})$.
3. Sender outputs $\sigma \leftarrow \sigma_{\mathsf{SoK}}$.

Verification protocol: Input is $(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \sigma, m)$

1. Parses σ as the SoK signature σ_{SoK}.
2. Verifies that σ_{SoK} is a valid SoK on the message m proving knowledge of $\mathsf{sk}_{L,i}$. I.e. it runs $b \leftarrow \mathsf{SoK.verify}((\mathbf{B}, \mathsf{sl}, \mathsf{R}), \sigma_{\mathsf{SoK}}, m)$.
3. Verifier outputs b.

Figure 7.2: AfP Protocol

Theorem 7.2 (EUF-CMA AfP). *Let $\mathcal{E}$ be an IND-CPA secure and $\mathbb{Z}_p$-linearly homomorphic encryption scheme and let* SoK *be a simulatable and extractable SoK scheme. The construction in Fig. 7.2 with lottery predicate as in Section 7.2.1 is EUF-CMA AfP as defined in Definition 6.5.*

(See proof sketch in Section 7.6)

7.2.3.1 AfP Privacy

The privacy property of an AfP scheme says that no adversary can distinguish between interacting with an AfP oracle $\mathcal{O}_{\mathsf{AfP}}$ and a simulator $\mathcal{S}$ during a blockchain execution. Intuitively, this provides the guarantee that observing other AfP tags does not enhance an adversary's chance of guessing future lottery winners.

Theorem 7.3 (AfP Privacy). *Assume $\mathcal{E}$,* lottery *and* SoK *scheme as in Theorem 7.5. The construction in Fig. 7.2 has AfP privacy as in Definition 6.6.*

(See proof sketch in Section 7.6). An AfP based on the setup presented in Fig. 7.1 will not provide a good foundation for YOSO-MPC or even just a proactive secret sharing scheme. The reason is, that as soon as a party $\mathcal{P}_i$ publishes an AfP tag, any other party can verify that $\mathcal{P}_i$ won the lottery and, thus, link the identity of $\mathcal{P}_i$ to the public key $\mathsf{pk}_{\mathsf{Anon},\psi(i)}$ from the output of the mixnet. This will ruin the setup for this party when future lotteries are conducted. More importantly, a powerful adversary is able to identify any subsequent ECW ciphertexts towards this party and can design its corruption strategy accordingly. What we want is a new ephemeral public key $\mathsf{pk}_{\mathsf{Anon},\psi(i)}$ for each party and for each slot sl in the blockchain execution where an AfP is produced. Note that a new lottery setup is

necessary for each slot sl even though different parties are producing AfP tags in different slots. The reason is that observing *any* AfP tag, inadvertently, skews the probability distribution and helps the adversary in guessing future lottery winner.
A simple way to solve the above issue is to repeat the lottery setup and obtain multiple vectors of the format $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$. Then, any party can use a new anonymized public key for each AfP tag. We describe this property as *bounded* AfP privacy. Bounded AfP privacy ensures that the AfP scheme can be executed a bounded number times before lottery winners can be linked to specific public keys in the setup and ECW ciphertexts starts betraying their receivers. Note that the idea of generating multiple lottery setups in batches (preprocessing) can result in more efficient protocols. But it has the downside that, while using the preprocessed public keys, the number of parties in the system is static.
In Section 7.5 we look at how to use the ECW and AfP in an anonymous PVSS protocol where we want encrypt towards multiple parties. In such a setting we can use linkable ring signatures (Section 7.9) to prove membership in a committee without directly revealing our public key in the setup.

7.2.4 AfP with Reusable Setup

In Section 7.9, we describe an efficient NIZK that allows for a party $\mathcal{P}_i$ to prove knowledge of a lottery secret key $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_j, \mathsf{sk}_{L,i}) = 1$ for $\mathsf{R}_j \in \{\mathsf{R}_1, \dots, \mathsf{R}_n\}$ without revealing R_j. Using this NIZK and an anonymous channel, we can construct an AfP that can be used multiple times without linking a party P_i to its setup public key. In order to generate an AfP on message m on behalf of role R in slot sl, P_i with $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i}) = 1$ first generates a NIZK π proving knowledge of $\mathsf{sk}_{L,i}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_j, \mathsf{sk}_{L,i}) = 1$ for $\mathsf{R}_j \in \{\mathsf{R}_1, \dots, \mathsf{R}_n\}$. Now P_i generates an SoK σ on the message m of knowledge of a valid proof π for the aforementioned statement. $\mathcal{P}_i$ publishes σ through an anonymous channel, avoiding its identity to be linked to the set $\{\mathsf{R}_1, \dots, \mathsf{R}_n\}$. The security and privacy guarantees for this AfP follow in a straightforward way from our previous analysis. While using this construction has a clear extra cost in relation to our simple AfP, we show in Section 7.9.2 how to efficiently perform such a reusable setup AfP on a set of ciphertexts, which is useful for our resharing application.

7.3 Publicly Verifiable Secret Sharing

7.3.1 Model

We define a publicly verifiable secret sharing (PVSS) scheme with t privacy and $t+1$-reconstruction, based on the models provided in [Sch99, RV05, HV09, CD17]. The goal is for a dealer to share a secret $S \in \mathbb{G}$ to a set of n parties $\mathcal{P} = \{P_1, \cdots, P_n\}$, so that $t+1$ shares will be needed to reconstruct the secret and no information will be revealed from t shares. We require public verifiability for correctness of sharing by the dealer, and for reconstruction of the secret by a set of $t+1$ parties. Due to this requirement, the protocol is entirely carried out using a public ledger.

We provide the syntax below. A modification we introduce with respect to the usual model is that we include asymmetric key pairs for dealers and an additional initial round where the parties can broadcast an ephemeral public key. This will allow for more efficient constructions as we will see in Section 7.3.3.

Setup

- $\mathsf{Setup}(1^\lambda)$ outputs public parameters pp
- $\mathsf{DKeyGen}(pp)$, performed by the dealer, outputs a key pair $(\mathsf{pk}_D, \mathsf{sk}_D)$
- $\mathsf{KeyGen}(pp, id_i)$, performed by i-th share receiver, outputs a key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$
- $\mathsf{VerifyKey}(pp, id, \mathsf{pk})$, performed by a public verifier, outputs 0/1 (as a verdict on whether pk is valid)

Distribution

- $\mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S)$ performed by the dealer, and where $S \in \mathbb{G}$ is a secret, outputs encrypted shares $C_i : i \in [n]$ and a proof $\mathsf{Pf}_{\mathsf{Sh}}$ of sharing correctness.

Verification

- $\mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$ performed by the public verifier outputs 0/1 (as a verdict on whether the sharing is valid)

Reconstruction

- $\mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$, performed by a share receiver, outputs a decrypted share A_i and a proof $\mathsf{Pf}_{\mathsf{Dec}i}$ of correct decryption.
- $\mathsf{VerifyDec}(pp, \mathsf{pk}_D, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i})$ outputs 0/1 (as a verdict on whether A_i is a valid decryption of C_i)
- $\mathsf{Rec}(pp, \{A_i : i \in \mathcal{T}\})$ for some $\mathcal{T} \subseteq [n]$ of size $t+1$ outputs a secret S. We will only apply this algorithm to inputs where $\mathcal{T}$ is of size $t+1$ and such that all A_i have passed the verification check.

We let $\mathcal{PK}_D$ and $\mathcal{PK}$ contain all key pairs output by $\mathsf{DKeyGen}$ and KeyGen respectively. For non–deterministic algorithms we sometimes explicitly reference the randomness r input. For example, $\mathsf{Dist}(pp, \mathsf{pk}_D, sk_D, \{\mathsf{pk}_i : i \in [n]\}, S; r)$. One of our constructions will not require pk_D, sk_D and consequently $\mathsf{DKeyGen}$. In that case we omit these arguments from the inputs to the other algorithms.

We require a PVSS to satisfy correctness, verifiability and IND1-secrecy.

Definition 7.1 (Correctness). *A PVSS satisfies* correctness *if for each secret $S \in \mathbb{G}$ and for*

any set of identifiers $\{id_i : i \in [n]\}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \mathsf{sk}_D) \leftarrow \mathsf{DKeyGen}(pp); \\ \forall i \in [n] \quad (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(pp, id_i); \\ (\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \leftarrow \\ \mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S); \\ \forall i \in [n] \quad (A_i, \mathsf{Pf}_{\mathsf{Dec}i}) \leftarrow \\ \mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i); \\ S' \leftarrow \mathsf{Rec}(pp, \{A_i : i \in [n]\}); \end{array} : \begin{array}{l} \forall i \in [n] \mathsf{VerifyKey}(pp, id_i, \mathsf{pk}_i) = 1 \\ \wedge \mathsf{Verify}(pp, \mathsf{pk}_D, \\ \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1 \\ \wedge \forall i \in [n] \mathsf{VerifyDec}(pp, \mathsf{pk}_D, \\ \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}i}) = 1 \\ \wedge S' = S \end{array}\right] = 1.$$

Verifiability The verifiability requirement ensures that an adversary must honestly follow the protocol. This means that it can be verified that parties honestly generate their ephemeral public keys (key generation), the dealer outputs encrypted shares for a secret (distribution), and that the parties honestly decrypt their shares in reconstruction (decryption).

Definition 7.2 (Verifiability of Key Generation)**.** *A PVSS satisfies* verifiability of key generation *if there exists a negligible function* $\mu(\lambda)$*such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, id, \mathsf{pk}) \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{l} \mathsf{VerifyKey}(pp, id, \mathsf{pk}) = 1 \\ \wedge \nexists \mathsf{sk} \ s.t. \ (\mathsf{sk}, \mathsf{pk}) \in \mathcal{PK} \end{array}\right]\right| \leq \mu(\lambda).$$

Definition 7.3 (Verifiability of Distribution)**.** *A PVSS satisfies* verifiability of distribution *if there exists a negligible function* $\mu(\lambda)$*such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \\ \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{l} \mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \\ = 1 \\ \wedge \nexists S, \mathsf{sk}_D, r \ s.t. \\ ((\mathsf{sk}_D, \mathsf{pk}_D) \in \mathcal{PK}_D \wedge \\ \mathsf{Dist}(pp, \mathsf{pk}_D, sk_D, t, \{\mathsf{pk}_i : i \in [n]\}, S; r) \\ = (\{C_i : i \in [n]\}, \cdot)) \end{array}\right]\right|$$

$$\leq \mu(\lambda).$$

Definition 7.4 (Verifiability of Decryption)**.** *A PVSS satisfies* verifiability of decryption *if there exists a negligible function* $\mu(\lambda)$*such that*

$$\left|\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda, t, n); \\ (\mathsf{pk}_D, \mathsf{pk}, C, A, \mathsf{Pf}_{\mathsf{Dec}}) \leftarrow \mathcal{A}(pp); \end{array} : \begin{array}{l} \mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}, C, A, \mathsf{Pf}_{\mathsf{Dec}}) = 1 \\ \wedge \nexists(\mathsf{sk}, r) \ s.t. ((\mathsf{sk}, \mathsf{pk}) \in \mathcal{PK} \wedge \\ \mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C; r) = (A, \cdot)) \end{array}\right]\right|$$

$$\leq \mu(\lambda).$$

Indistinguishability of Secrets (IND-1 Secrecy) We now present the IND-1 Secrecy definition from [CD17]. We have modified this definition to fit the adjusted syntax because Dist can no longer be performed by the adversary (as it takes sk_D as input). We now provide a DIST oracle that will return the outputs of the Dist algorithm. To capture that the public keys of the parties should be ephemeral, we do not allow the public keys of parties that are used in the challenge to be input to this oracle. We therefore allow the adversary to output an extra $n - k$ keys.

Algorithm 16: $\mathsf{Game}^{\text{ind-secrecy},b}_{\mathcal{A},\text{PVSS}}$

procedure DIST($(\mathcal{U}, S')$)
  if $\mathcal{U} \not\subseteq [n+1, k]$ or $|\mathcal{U}| \neq n$ **then**
    return $\perp$
  end if
  $(\{C'_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \leftarrow \mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, t, n, \{\mathsf{pk}_i : i \in \mathcal{U}\}, S')$
  return $(\{C'_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$
end procedure
procedure $\mathsf{Game}^{\text{ind-secrecy},b}_{\mathcal{A},\text{PVSS}}(\lambda)$
  $pp \leftarrow \mathsf{Setup}(1^\lambda, t, n)$, $(\mathsf{pk}_D, \mathsf{sk}_D) \leftarrow \mathsf{DKeyGen}(pp)$
  $\forall i \in [n-t] \quad (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(pp, i)$
  $(\{\mathsf{pk}_i : i \in [n-t+1, n]\}, \{\mathsf{pk}_i : i \in [n+1, k]\}) \leftarrow \mathcal{A}(pp, \mathsf{pk}_D, \{\mathsf{pk}_i : i \in [n-t]\})$
  if $\exists i \in [n-t+1, k]$ such that $\mathsf{VerifyKey}(pp, i, \mathsf{pk}_i) = 0$ **then**
    return 0
  end if
  $S_0, S_1 \leftarrow\$ \mathbb{G}$, $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) \leftarrow \mathsf{Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, t, \{\mathsf{pk}_i : i \in [n]\}, S_0)$
  $b' \leftarrow \mathcal{A}^{\mathsf{DIST}}(S_b, \{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$
  return b'
end procedure

Definition 7.5 (IND-1 Secrecy). *A PVSS satisfies* indistinguishability of secrets *if, for any PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\lambda)$such that*

$$\left|\Pr\left[\mathsf{Game}^{\text{ind-secrecy},0}_{\mathcal{A},\text{PVSS}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\text{ind-secrecy},1}_{\mathcal{A},\text{PVSS}}(\lambda) = 1\right]\right| \leq \mu(\lambda).$$

7.3.2 HEPVSS: Generic PVSS from $\mathbb{Z}_p$-LHE Scheme

We present in Fig. 7.3 our construction for a PVSS scheme HEPVSS based on a $\mathbb{Z}_p$-LHE scheme with proof of correct decryption. This construction does not require the dealer to hold a key pair or parties to prove honest generation of keys and therefore we remove this from the syntax. Moreover, because the dealer does not have a key pair, here we do not require the public keys pk_i to be ephemeral.

The construction is relatively straightforward: the dealer construct the (group) Shamir sharing of the secret, and encrypts the shares using the $\mathbb{Z}_p$-LHE scheme, resulting in cyphertexts C_i. The sharing correctness proof needs to assert, not only that each C_i is

individually a correct encryption, but also that the underlying plaintext messages are evaluations of a polynomial of degree at most t. Here we use the fact that the set of polynomials of degree at most t is a vector space, and the map that sends a polynomial to its evaluation in some point is linear, so we can capture the above statement in terms of knowledge of preimage of a certain linear map.

Algorithms for HEPVSS

HEPVSS.Setup($1^\lambda, t, n$):

1: $(\mathbb{G}, G, p, \mathcal{E}) \leftarrow\$ \mathcal{G}(1^\lambda)$. Choose pairwise distinct $\alpha_0, \alpha_1, \cdots \alpha_n \in \mathbb{Z}_p$
2: **return** $pp = (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\}, \mathcal{E})$

HEPVSS.KeyGen(pp, id):

1: **return** $(\mathsf{sk}, \mathsf{pk}) \leftarrow\$ \mathcal{E}.g(1^\lambda)$

HEPVSS.Dist($pp, \{\mathsf{pk}_i : i \in [n]\}, S$):

1: Parse pp as $(\mathbb{G}, G, p, n, \{\alpha_i : i \in [0, n]\}, \mathcal{E}) := (pp_{\mathsf{Sh}}, \mathcal{E})$
2: $(\{A_i : i \in [n]\}, m(X)) \leftarrow \mathsf{GShamir}(pp_{\mathsf{Sh}}, S)$
3: **for** $i \in [n]$ **do**
4: $\quad \rho_i \leftarrow\$ \mathfrak{R}$, $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(A_i, \rho_i)$
5: **end for**
6: $\mathcal{W} \leftarrow \mathbb{G} \times \mathbb{Z}_p[X]_{\leq t} \times \mathfrak{R}^n, \quad \mathcal{X} \leftarrow \{0\} \times \mathfrak{C}^n, \quad pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
7: $w \leftarrow (S, m(X), \rho_1, \dots, \rho_n), \quad x \leftarrow (0, C_1, \dots, C_n)$
8: Let f given by
9: $\quad f(w) := (m(\alpha_0), \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_1}(S + m(\alpha_1) \cdot G; \rho_1), \dots, \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_n}(S + m(\alpha_n) \cdot G; \rho_n))$
10: $\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp_\pi, x, f)$
11: **return** $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

HEPVSS.Verify($pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}$):

1: **return** $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, x, f, \mathsf{Pf}_{\mathsf{Sh}})$, with $\mathcal{W}, \mathcal{X}, pp_\pi, x, f$ as in HEPVSS.Dist

HEPVSS.DecShare($pp, \mathsf{pk}, \mathsf{sk}, C$):

1: $A \leftarrow \mathsf{Dec}_{\mathsf{sk}}(C)$, $\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathcal{E}.\mathsf{ProveDec}(A, C, \mathsf{pk})$
2: **return** $(A, \mathsf{Pf}_{\mathsf{Dec}})$

HEPVSS.VerifyDec($pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}\,i}$):

1: **return** $\mathcal{E}.\mathsf{VerifyDec}(A_i, C_i, \mathsf{pk}_i, \mathsf{Pf}_{\mathsf{Dec}\,i})$

HEPVSS.Rec($pp, \{A_i : i \in \mathcal{T}\}$):

1: **return** $\mathsf{GShamir.Rec}(pp, \{A_i : i \in \mathcal{T}\})$

Figure 7.3: Algorithms for Public Verifiable Secret Sharing Scheme HEPVSS

7.3.2.1 Security

We prove that HEPVSS satisfies correctness, indistinguishability of secrets and verifiability in Section 7.7.1.

7.3.3 DHPVSS: A PVSS with Constant-Size Sharing Correctness Proof

We now give an optimized construction of a PVSS with a proof of sharing correctness consisting of just two field elements. The PVSS scheme, which we call DHPVSS, has IND1-secrecy under the DDH assumption.

We explain the idea of the construction next: Let $A_i = a_i \cdot G$ be (purportedly) group Shamir shares for a secret $S \in \mathbb{G}$. A SCRAPE check (Theorem 5.1) consists on the verification $\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot a_i \stackrel{?}{=} 0$, or alternatively

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot A_i \stackrel{?}{=} O,$$

for O the identity element of $\mathbb{G}$. Here v_i are fixed coefficients dependent on the α_i and $m^*(X)$ is sampled uniformly at random from $\mathbb{Z}_p[X]_{\leq n-t-1}$. If it is not true that all a_i are of the form $m(\alpha_i)$ for some polynomial $m(X) \in \mathbb{Z}_p[X]_{\leq t}$, then the check succeeds with probability at most $1/p$.

In [CD17], the encrypted shares were $C_i = a_i \cdot \mathsf{pk}_i$. Because these are in different bases the check above cannot be directly applied on the C_i, and then the strategy consisted on sending additional elements $a_i \cdot H$ (for some group generator H), proving that the underlying a_i's are the same, and carrying out the check on these $a_i \cdot H$. All this introduces overhead which is linear in n.

Instead, in DHPVSS, the dealer has a key-pair $(\mathsf{sk}_D, \mathsf{pk}_D)$, with $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$, and encrypts A_i as $C_i = A_i + \mathsf{sk}_D \cdot E_i$, where $E_i = \mathsf{sk}_i \cdot G$ is an ephemeral public key of the i-th party. Note that $\mathsf{sk}_D \cdot E_i$ can be seen as a shared Diffie-Hellman key between dealer and the i-th party or, alternatively, C_i can be seen as an El-Gamal encryption of A_i under E_i with randomness sk_D.

The advantage is that now $\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot A_i \stackrel{?}{=} O$ is equivalent to

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot C_i \stackrel{?}{=} \mathsf{sk}_D \cdot \left(\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot E_i \right),$$

which is *one single* DLEQ proof $\mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$ for *publicly computable*

$$U = \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot E_i, \quad V = \sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot C_i.$$

One detail is that, as opposed to the PVSS in [CD17] (where $m^*(X)$ was locally sampled by the verifier), the prover needs to know $m^*(X)$ so this is sampled via a random oracle. The algorithms can be found in Fig. 7.4 and Fig. 7.5.

Algorithms for PVSS scheme DHPVSS**, Setup and Distribution**

DHPVSS.Setup($1^\lambda, t, n$):

1: $(\mathbb{G}, G, p) \leftarrow\$ \mathcal{G}(1^\lambda)$. Choose pairwise distinct $\alpha_0, \alpha_1, \cdots \alpha_n \in \mathbb{Z}_p$
2: $\forall i \in [n] \quad v_i \leftarrow \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}$
3: **return** $pp = (\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$

DHPVSS.DKeyGen(pp):

1: $\mathsf{sk}_D \leftarrow\$ \mathbb{Z}_p^*, \mathsf{pk}_D \leftarrow \mathsf{sk}_D \cdot G$
2: **return** $(\mathsf{pk}_D, \mathsf{sk}_D)$

DHPVSS.KeyGen(pp, id):

1: $\mathsf{sk} \leftarrow\$ \mathbb{Z}_p^*, E \leftarrow \mathsf{sk} \cdot G, \Omega \leftarrow \mathsf{DL}(\mathsf{sk}; G, E, id), \mathsf{pk} \leftarrow (E, \Omega)$
2: **return** $(\mathsf{pk}, \mathsf{sk})$

DHPVSS.VerifyKey(pp, id, pk):

1: **parse** pk as (E, Ω)
2: **return** accept iff Ω is valid w.r.t G, E, id

DHPVSS.Dist($pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S$):

1: **parse** pk_i as (E_i, Ω_i), pp as $(\mathbb{G}, G, p, t, n, \alpha_0, \{(\alpha_i, v_i) : i \in [n]\})$
2: $pp_{\mathsf{Sh}} \leftarrow (\mathbb{G}, G, p, t, n, \{\alpha_i : i \in [0, n]\})$
3: $(\{A_i\}_{i \in [n]}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathsf{Sh}}, S)$
4: $\forall i \in [n], C_i \leftarrow \mathsf{sk}_D \cdot E_i + A_i$
5: $m^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
6: $V \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i \cdot m^*(\alpha_i) \cdot E_i$
7: $\mathsf{Pf}_{\mathsf{Sh}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}_D; G, \mathsf{pk}_D, U, V)$
8: **return** $(\{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

Figure 7.4: Algorithms for PVSS scheme DHPVSS, Setup and Distribution

Algorithms for PVSS scheme DHPVSS**, Verification and Reconstruction**

DHPVSS.Verify($pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}$):

1: **parse** pk_i as (E_i, Ω_i), pp as $(\mathbb{G}, G, p, t, n, \{(\alpha_i, v_i) : i \in [n]\})$
2: $m^* \leftarrow \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
3: $V \leftarrow \sum_{i=1}^n v_i m^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^n v_i m^*(\alpha_i) \cdot E_i$
4: **return** accept iff $\mathsf{Pf}_{\mathsf{Sh}}$ is valid w.r.t G, pk_D, U, V

DHPVSS.DecShare($pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C$):

1: **parse** pk as (E, Ω)
2: $A' \leftarrow C - \mathsf{sk} \cdot \mathsf{pk}_D$
3: $\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathsf{DLEQ}(\mathsf{sk}; G, E, \mathsf{pk}_D, C - A')$
4: **return** $(A', \mathsf{Pf}_{\mathsf{Dec}})$

DHPVSS.VerifyDec($pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}\,i}$):

1: **parse** pk_i as (E_i, Ω_i)
2: **return** accept iff $\mathsf{Pf}_{\mathsf{Dec}\,i}$ is valid w.r.t $G, E_i, \mathsf{pk}_D, C_i - A_i$

DHPVSS.Rec($pp, \{A_i : i \in \mathcal{T}\}$):

1: **return** $\mathsf{GShamir.Rec}(pp, \{A_i : i \in \mathcal{T}\})$

Figure 7.5: Algorithms for PVSS scheme DHPVSS, Verification and Reconstruction

7.3.3.1 Security

We prove that DHPVSS satisfies correctness, indistinguishability of secrets and verifiability in Section 7.7.2.

7.3.3.2 Communication Complexity Comparison.

The communication complexity of DHPVSS.Dist is $(n+2)\log p$ bits. In contrast, HEPVSS.Dist instantiated with El Gamal is of $(3n+3)\log p$ bits. Secret distribution in SCRAPE [CD17] requires $(3n+1)\log p$ bits, which was reduced to $(n+t+2)\log p$ bits in ALBATROSS [CD20]. Therefore DHPVSS.Dist obtains an additive saving of $t\log p$ bits with respect to the best previous alternative. The communication of both DHPVSS.DecShare and HEPVSS.DecShare is $3\log p$ bits. The share decryption complexities in [CD17] and [CD20] are similar to ours. See Section 7.8 for more details.

7.4 PVSS Resharing

In this section we introduce protocols that allow a committee $\mathcal{C}_r$ of size n_r, among which a secret has been PVSSed with an underlying t_r-threshold Shamir scheme, to create a PVSS of the same secret for the next committee $\mathcal{C}_{r+1}$ of size n_{r+1} and with threshold t_{r+1}. By design, the protocols will keep the secret hidden from any adversary corrupting at most t_r parties from $\mathcal{C}_r$ and t_{r+1} from $\mathcal{C}_{r+1}$, and will be correct as long as there are t_r+1 honest parties in $\mathcal{C}_r$. In particular, this can be used by a party R to transmit a message to a committee in the future, by keeping this secret being reshared among successive committees and setting the last Shamir threshold to be 0.

Suppose for now that the secret sharing scheme were for secrets over $\mathbb{Z}_p$. Each party in $\mathcal{C}_r$ would hold $\sigma_\ell = m_r(\alpha_\ell)$ where m_r is the sharing polynomial for that round, of degree t_r. A subcommittee L_r of t_r+1 parties in $\mathcal{C}_r$ can then reshare the secret by PVSSing their shares among $\mathcal{C}_{r+1}$ with Shamir scheme of degree t_r+1. The parties in $\mathcal{C}_{r+1}$ then compute the sum of the received shares weighted by coefficients

$$\lambda_{\ell,L_r} := \prod_{j\in L_r, j\neq\ell} \frac{\alpha_0-\alpha_j}{\alpha_\ell-\alpha_j}$$

Indeed, if we denote $[\sigma_\ell]$ the vector of shares sent by P_ℓ in L_r, then

$$\sum_{\ell\in L_r} \lambda_{\ell,L_r}[\sigma_\ell] = \sum_{\ell\in L_r} \lambda_{\ell,L_r}[m(\alpha_\ell)] = [\sum_{\ell\in L_r} \lambda_{\ell,L_r} m(\alpha_\ell)] = [m(\alpha_0)]$$

In our situation, each party $R_{r,i}$ in $\mathcal{C}_r$ has instead a group element as share, and needs to PVSS it among $\mathcal{C}_{r+1}$ using the algorithm Dist from previous section. However, the proof in Dist only guarantees that the distributed shares are consistent with some secret. Here we require in addition that this secret is the shared that the party has received previously.

To be more precise, in round r, each party $\mathsf{R}_{\mathsf{r},i}$ in committee $\mathcal{C}_\mathsf{r}$ has $A_{\mathsf{r},i}$ as share and in addition the encryption $C_{\mathsf{r},i} = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{r},i}}(A_{\mathsf{r},i})$ of $A_{\mathsf{r},i}$ is public. $\mathsf{R}_{\mathsf{r},i}$ now needs to create shares of $A_{\mathsf{r},i}$ for the committee $C_{\mathsf{r}+1}$. Let $A_{i\to j}$ be the share that will be sent to $\mathsf{R}_{\mathsf{r}+1,j}$. This will be encrypted as $C_{i\to j} = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{r}+1,j}}(A_{i\to j})$ and $\mathsf{R}_{\mathsf{r},i}$ must prove that $C_{i\to j}$ are encryptions of a correct sharing whose secret is indeed the plaintext of $C_{\mathsf{r},i}$.

When a subset L_r of $\mathcal{C}_\mathsf{r}$ of $t_\mathsf{r}+1$ parties have correctly reshared, each $\mathsf{R}_{\mathsf{r}+1,j}$ sets $A_{\mathsf{r}+1,j} = \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} A_{\ell\to j}$ as their share and everyone sets $C_{\mathsf{r}+1,j} = \sum_{\ell\in L_\mathsf{r}} \lambda_{\ell,L_\mathsf{r}} C_{\ell\to j}$ as the corresponding public ciphertext for that share.

7.4.1 Resharing for HEPVSS

In the case of HEPVSS, the additional proof that the reshared value is the one corresponding to the public ciphertext can be integrated easily in HEPVSS.Dist if the encryption scheme has $\mathbb{Z}_p$-linear decryption.

Let $\mathsf{pk}_{[n]}$ denote the set $\{\mathsf{pk}_i : i \in [n]\}$. Similarly $C_{[n]}$ denote a set of ciphertexts $\{C_i : i \in [n]\}$ and $\rho_{[n]}$ denote a set of elements from the randomness space $\{\rho_i : i \in [n]\}$. Recall $D_C(\mathsf{sk}) := \mathsf{Dec}_{\mathsf{sk}}(C)$. Define the relation

$$\begin{aligned} R_{\mathsf{Reshare}} = &\{(m(X), \mathsf{sk}, \rho_{[n]}); (\mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}) : \\ &F(\mathsf{sk}) = \mathsf{pk}, m(\beta_0) = 0, \mathsf{Enc}_{\mathsf{pk}_i}(m(\beta_i)\cdot G + D_C(\mathsf{sk}); \rho_i) = C_i \text{ for } i \in [n]\} \end{aligned}$$

We therefore define the resharing proof in Fig. 7.6. The protocol for PVSS resharing is then constructed as in Fig. 7.7.

Proof HEPVSS.Reshare **of correct resharing of encrypted secret**

HEPVSS.Reshare.Prove$((m(X), \mathsf{sk}, \rho_{[n]}); (pp, \mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}))$:

1: **parse** $pp = (\mathbb{G}, G, p, t, n, \{\beta_i : i \in [n]\})$
2: $\mathcal{W} \leftarrow \mathbb{Z}_p[X]_{\le t} \times \mathcal{SK} \times \mathfrak{R}^n$, $\mathcal{X} \leftarrow \mathcal{PK} \times \mathfrak{C}^n$,
3: $pp' \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$, $w \leftarrow (m(X), \mathsf{sk}, \rho_1, \dots, \rho_n)$, $x \leftarrow (0, \mathsf{pk}, C_1, \dots, C_n)$,
4: Set f_C given by $f_C(w) := (m(\beta_0), F(\mathsf{sk}), \mathsf{Enc}_{pk_1}(A_1; \rho_1), \dots, \mathsf{Enc}_{pk_1}(A_n; \rho_n))$
5: where $A_i = m(\beta_i)\cdot G + D_C(\mathsf{sk})$
6: **return** $\Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp', x, f_C)$

HEPVSS.Reshare.Verify$(pp, \mathsf{pk}, \mathsf{pk}_{[n]}, C, C_{[n]}, \pi)$:

1: Set $\mathcal{W}, \mathcal{X}, pp', x, f_C$, as in Reshare.Prove
2: **return** $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp', x, f_C, \pi)$

Figure 7.6: Proof system HEPVSS.Reshare for correct resharing of encrypted secret

Protocol for HEPVSS **PVSS resharing**

Participants: Disjoint committees $\mathcal{C}_r = \{P_{r,1}, \dots, P_{r,n_r}\}$ and $\mathcal{C}_{r+1} = \{P_{r+1,1}, \dots, P_{r+1,n_{r+1}}\}$.

Public information: A group $\mathbb{G}$ of prime order p, with generator G. A homomorphic encryption scheme $\mathcal{E}$ with $\mathbb{Z}_p$-linear decryption, with plaintext space $\mathbb{G}$. Public keys $\mathsf{pk}_{j,i}$ for that encryption scheme corresponding to parties $\mathsf{R}_{j,i}$ above ($j = r, r+1, 1 \le i \le n_r$), where $\mathsf{R}_{j,i}$ knows the corresponding secret key $\mathsf{sk}_{j,i}$; thresholds t_r, t_{r+1}. Evaluation points $(\alpha_0, \alpha_1, \dots, \alpha_{n_r})$, $(\beta_0, \beta_1, \dots, \beta_{n_{r+1}})$.
Input: Public ciphertexts $C_{r,i}$, where it is guaranteed that $C_{r,i} = \mathsf{Enc}_{\mathsf{pk}_{r,i}}(A_{r,i})$ such that $A_{r,i} = f_r(\alpha_i) \cdot G$ for some polynomial f_r of degree $\le t_r$.

Output: A public output $(C_{r+1,1}, \dots, C_{r+1,n_{r+1}})$ and a proof π that, for all $k = 1, \dots, n_{r+1}$, $C_{r+1,k} = \mathsf{Enc}_{\mathsf{pk}_{r+1,i}}(A_{r+1,k})$ such that $A_{r+1,k} = f_{r+1}(\beta_k) \cdot G$ for some polynomial f_{r+1} of degree $\le t_{r+1}$ and $f_{r+1}(\beta_0) = f_r(\alpha_0)$.

Protocol:

1. Let $pp_{r+1} = (\mathbb{G}, G, p, t_{r+1}, n_{r+1}, \{\beta_i : i \in [0, n]\})$.
2. Resharing: For $i = 1, \dots, n_r$, $P_{r,i}$ does the following
   a) Compute $A_{r,i} = \mathsf{Dec}_{\mathsf{sk}_{r,i}}(C_{r,i})$.
   b) For $i = 1, \dots, n_r$, $P_{r,i}$ computes
   $(\{A_{i\to j} : j \in [n_{r+1}]\}, m(X)) \leftarrow \mathsf{GShamir.Share}(pp_{r+1}, A_{r,i})$
   c) Sample $\rho_{i\to j} \in \mathfrak{R}$ for $j \in [n_{r+1}]$ and let $\rho_{i\to[n_{r+1}]} = \{\rho_{i\to j} : j \in [n_{r+1}]\}$
   d) Compute $C_{i\to j} = \mathsf{Enc}_{\mathsf{pk}_j}(A_{i\to j}; \rho_{i\to j})$ for $j \in [n_{r+1}]$.
   e) Compute
   $$\pi_i \leftarrow \mathsf{HEPVSS.Reshare.Prove}\quad (m(X), \mathsf{sk}, \rho_{i\to[n_{r+1}]}; pp, \mathsf{pk}_{r,i}, \{\mathsf{pk}_{r+1,j} : j \in [n_{r+1}]\}, C_{r,i}, \{C_{i\to j} : j \in [n_{r+1}]\})$$
   f) Output $\{C_{i\to j} : j \in \mathcal{C}_{r+1}\}, \pi_i$
3. Reconstruction of next share encryptions: each party in $\mathcal{P}$ locally constructs the encryptions of the shares for the following round as follows:
   a) Define L containing the first $t+1$ indices i for which the following accepts:
   $$\mathsf{HEPVSS.Reshare.Verify}(\quad pp, \mathsf{pk}_{r,i}, \{\mathsf{pk}_{r+1,j} : j \in [n_{r+1}]\}, C_{r,i}, \{C_{i\to j} : j \in [n_{r+1}]\}, \pi_i)$$
   b) For $j \in [n_{r+1}]$, set $C_{r+1,j} = \sum_{\ell \in L} \lambda_{\ell,L} C_{\ell\to j}$ [a]
   c) Output $\{(C_{r+1,j} : j \in [n_{r+1}]\}, (\pi_{r,\ell})_{\ell \in L})$.

[a] Here $\sum$ refers to the summatory with respect to the homomorphic operation on ciphertexts $\boxplus_{\mathfrak{C}}$

Figure 7.7: Protocol for HEPVSS resharing

7.4.2 Resharing for DHPVSS

In the case of DHPVSS, the situation is slightly more complicated due to the fact that the encryption of shares involves a key from the dealer. Here there are different dealers, i.e. the final share of each party in $\mathcal{C}_{r+1}$ is a linear combination of shares sent by the parties in L_r. Thanks to the fact that the encryption is also a linear operation with respect to the public key of the sender, we can define a public key for committee L_r. Indeed, if we call pk_{D_ℓ} the public key of $P_{r,\ell}$ when acting as sender, then $\mathsf{pk}_{D,L_r} := \sum_{\ell \in L_r} \lambda_{\ell,L_r} \cdot \mathsf{pk}_{D_\ell}$. Then we want to make sure that the output encryption for $P_{r+1,j}$ is

$$C_{r+1,j} = \mathsf{sk}_{r+1,j} \cdot \mathsf{pk}_{D,L_r} + \sum_{\ell \in L_r} \lambda_{\ell,L_r} A_{\ell \to j}.$$

At the beginning of the resharing, each party $P_{r,i}$ in committee $\mathcal{C}_r$ has as share $A_{r,i} = C_{r,i} - \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{r-1}}$ where sk_i is the secret key for decrypting shares, and needs to create shares $A_{i \to j}$ of $A_{r,i}$ and encrypt them using the public keys $\mathsf{pk}_{[n_{r+1}]} = \{\mathsf{pk}_j : j \in [n_{r+1}]\}$ of the parties of the next round and its own secret key sk_{D_i} (i.e. this party will create $C_{[n_{r+1}]} = \{C_{i \to j} : j \in [n_{r+1}]\}$ with $C_{i \to j} = \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j + A_{i \to j}$) and prove their validity. In conclusion we need a proof for the following relation

$$\begin{aligned} R_{\mathsf{DHPVSS,Reshare}} = \{ & (m(X), \mathsf{sk}_i, \mathsf{sk}_{D_i}); (pp, \mathsf{pk}_i, \mathsf{pk}_{D_i}, \mathsf{pk}_{D,L_{r-1}}, \mathsf{pk}_{[n_{r+1}]}, C_{r,i}, C_{[n_{r+1}]}) : \\ & \mathsf{pk}_i = \mathsf{sk}_i \cdot G,\ \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G,\ m(X) \in \mathbb{Z}_p[X]_{\leq t},\ m(\beta_0) = 0, \\ & \text{and } \forall j \in [n_{r+1}],\ C_{i \to j} = \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j + A_{i \to j}, \\ & \text{where } A_{i \to j} = (C_{r,i} - \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{r-1}}) + m(\beta_j) \cdot G \} \end{aligned}$$

However, we also want to use the SCRAPE technique to reduce the size of the witness and hence of the proof. Note that if we set

$$U_j = C_{i \to j} - \mathsf{sk}_{D_i} \cdot \mathsf{pk}_j - C_{r,i} + \mathsf{sk}_i \cdot \mathsf{pk}_{D,L_{r-1}}$$

for all $j \in [n_{r+1}]$ and $U_0 = O$, we want to make sure that for all $j \in [0, n_{r+1}]$, $U_j = m(\beta_j) \cdot G$ for a polynomial of degree $\leq t$ (in addition to the conditions $\mathsf{pk}_i = \mathsf{sk}_i \cdot G$ and $\mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G$).

For $j \in [0, n]$, let

$$v'_j = \prod_{k \in [0,n] \setminus \{j\}} (\beta_j - \beta_k)^{-1}.$$

Observe these are not exactly the same coefficients as in the description of DHPVSS in Section 7.3.3 because they include the evaluation point β_0. By Theorem 5.1, we want to prove $\sum_{j=0}^{n} v'_i \cdot m^*(\beta_j) \cdot U_j = O$, for a random polynomial m^* of degree $n - t$.

Observe $\sum_{j=0}^{n} v'_j \cdot m^*(\beta_j) \cdot U_j = U' - \mathsf{sk}_{D_i} \cdot V' + \mathsf{sk}_i \cdot W'$ for publicly computable

$$U' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot (C_{i \to j} - C_{r,i}),\ V' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot \mathsf{pk}_j,\ W' := \sum_{j=1}^{n} v'_j \cdot m^*(\beta_j) \cdot \mathsf{pk}_{D,L_{r-1}},$$

and therefore $P_{r,i}$ needs a proof of knowledge for

$$R'_{\mathsf{DHPVSS},\mathsf{Reshare},m^*} = \{(\mathsf{sk}_i, \mathsf{sk}_{D_i}); (\mathsf{pk}_i, \mathsf{pk}_{D_i}, U', V', W') : \\ \mathsf{pk}_i = \mathsf{sk}_i \cdot G, \quad \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G, \quad U' = \mathsf{sk}_{D_i} \cdot V' - \mathsf{sk}_i \cdot W'\}$$

where we remark that now the witness only contains two elements but on the other hand the relation depends on a polynomial $m^*(X)$ that has been sampled uniformly at random among polynomials of degree at most $n - t$. This leads to the protocol for PVSS resharing in Fig. 7.8.

Protocol for DHPVSS **resharing**

Participants: $\mathcal{C}_{\mathsf{r}} = \{P_{\mathsf{r},1}, \ldots, P_{\mathsf{r},n_{\mathsf{r}}}\}$ and $\mathcal{C}_{\mathsf{r}+1} = \{P_{\mathsf{r}+1,1}, \ldots, P_{\mathsf{r}+1,n_{\mathsf{r}+1}}\}$.

Public information: A group $\mathbb{G}$ of prime order p, with generator G. "Sender" key pairs $(\mathsf{sk}_{D_i}, \mathsf{pk}_{D_i} = \mathsf{sk}_{D_i} \cdot G)$ for every party $\mathsf{R}_{\mathsf{r},i} \in \mathcal{C}_{\mathsf{r}}$, a "sender committee" public key $\mathsf{pk}_{D,L_{\mathsf{r}-1}}$, and "receiver" key pairs $(\mathsf{sk}_{r,i}, \mathsf{pk}_{r,i} = \mathsf{sk}_{r,i} \cdot G)$ for $\mathsf{R}_{r,i}$, where $r = \mathsf{r}, \mathsf{r}+1$, and $1 \leq i \leq n_r$; thresholds t_{r}, $t_{\mathsf{r}+1}$. Evaluation points $(\alpha_0, \alpha_1, \ldots, \alpha_{n_{\mathsf{r}}})$, $(\beta_0, \beta_1, \ldots, \beta_{n_{\mathsf{r}+1}})$. Random oracles $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p[X]_{\leq n-t}$, $\mathcal{H}' : \{0,1\}^* \to \mathbb{Z}_p$. Let $\mathcal{W} \leftarrow \mathbb{Z}_p^2$, $\mathcal{X} \leftarrow \mathbb{G}^3$, and $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H}')$.

Input: Public ciphertexts $C_{\mathsf{r},i} = \mathsf{sk}_{\mathsf{r},i} \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}} + A_{\mathsf{r},i}$ such that $A_{\mathsf{r},i} = h_{\mathsf{r}}(\alpha_i) \cdot G$ for some polynomial h_{r} of degree $\leq t_{\mathsf{r}}$.

Output: A public key $\mathsf{pk}_{D,L_{\mathsf{r}}}$ for a subset L_{r} of $\mathcal{C}_{\mathsf{r}}$, of size $t_{\mathsf{r}}+1$. Public output ciphertexts $(C_{\mathsf{r}+1,1}, \ldots, C_{\mathsf{r}+1,n_{\mathsf{r}+1}})$ and a proof π that, for all $j = 1, \ldots, n_{\mathsf{r}+1}$, $C_{\mathsf{r}+1,j} = \mathsf{sk}_{\mathsf{r}+1,j}\mathsf{pk}_{D,L_{\mathsf{r}}} + A_{\mathsf{r}+1,j}$ such that $A_{\mathsf{r}+1,j} = h_{\mathsf{r}+1}(\beta_j) \cdot G$ for some polynomial $h_{\mathsf{r}+1}$ of degree $\leq t_{\mathsf{r}+1}$ and $h_{\mathsf{r}+1}(\beta_0) = h_{\mathsf{r}}(\alpha_0)$.

Protocol:

1. Let $pp_{\mathrm{Sh},\mathsf{r}+1} = (\mathbb{G}, G, p, t_{\mathsf{r}+1}, n_{\mathsf{r}+1}, \{\beta_j : j \in [0, n_{\mathsf{r}+1}]\})$.
2. Resharing: For $i = 1, \ldots, n_{\mathsf{r}}$, $P_{\mathsf{r},i}$ does the following:
   a) $A_{\mathsf{r},i} \leftarrow C_{\mathsf{r},i} - \mathsf{sk}_{\mathsf{r},i} \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$.
   b) $(\{A_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, m_i(X)) \leftarrow \mathsf{GShamir.Share}(pp_{\mathrm{Sh},\mathsf{r}+1}, A_{\mathsf{r},i})$.
   c) For $j \in [n_{\mathsf{r}+1}]$, $C_{i\to j} \leftarrow \mathsf{sk}_{D_i} \cdot \mathsf{pk}_{\mathsf{r}+1,j} + A_{i\to j}$.
   d) $m_i^*(X) \leftarrow \mathcal{H}(\{C_{\mathsf{r},i} : i \in [n_{\mathsf{r}}]\}, \mathsf{pk}_{D,L_{\mathsf{r}-1}})$.
   e) $U_i' \leftarrow \sum_{j=1}^{n} v_j' \cdot m_i^*(\beta_j) \cdot (C_{i\to j} - C_{\mathsf{r},i})$, $V_i' \leftarrow \sum_{j=1}^{n} v_j' \cdot m_i^*(\beta_j) \cdot \mathsf{pk}_{\mathsf{r}+1,j}$, $W_i' \leftarrow (\sum_{j=1}^{n} v_j' \cdot m_i^*(\beta_j)) \cdot \mathsf{pk}_{D,L_{\mathsf{r}-1}}$.
   f) $\pi_{\mathsf{r},i} \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}((\mathsf{sk}_{\mathsf{r},i}, \mathsf{sk}_{D_i}); pp_\pi, (\mathsf{pk}_{\mathsf{r},i}, \mathsf{pk}_{D_i}, U_i'), f_i)$, where $f_i(\mathsf{sk}_{\mathsf{r},i}, \mathsf{sk}_{D_i}) := (\mathsf{sk}_{\mathsf{r},i} \cdot G, \mathsf{sk}_{D_i} \cdot G, \mathsf{sk}_{D_i} \cdot V_i' - \mathsf{sk}_{\mathsf{r},i} \cdot W_i')$.
   g) Output $\{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}, \pi_{\mathsf{r},i}$.
3. Reconstruction of next share encryptions: each party in $\mathcal{P}$ locally constructs the encryptions of the shares for the following round as follows:
   a) For each $i \in \mathcal{C}_{\mathsf{r}}$:
      i. Compute U_i' and f_i as above (from public information and $P_{\mathsf{r},i}$'s output $\{C_{i\to j} : j \in [n_{\mathsf{r}+1}]\}$).
      ii. Compute $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, (\mathsf{pk}_{\mathsf{r},i}, \mathsf{pk}_{D_i}, U_i'), f_i, \pi_{\mathsf{r},i})$.
   b) Define L_{r} the set of $t+1$ first indices for which the above proofs accept.
   c) For $j \in [n_{\mathsf{r}+1}]$, $C_{\mathsf{r}+1,j} \leftarrow \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L} \cdot C_{\ell\to j}$.
   d) $\mathsf{pk}_{D,L_{\mathsf{r}}} \leftarrow \sum_{\ell \in L_{\mathsf{r}}} \lambda_{\ell,L_{\mathsf{r}}} \cdot \mathsf{pk}_{D_\ell}$.
   e) Output $(\{C_{\mathsf{r}+1,j} : j \in [n_{\mathsf{r}+1}]\}, (\pi_{\mathsf{r},\ell})_{\ell \in L_{\mathsf{r}}}, \mathsf{pk}_{D,L_{\mathsf{r}}})$.

Figure 7.8: Protocol for DHPVSS resharing

7.5 Anonymous PVSS via ECW and AfP

In this section, we show how to construct PVSS (and re-sharing) for anonymous committees by instantiating our previous PVSS constructions using our ECW and AfP schemes. We start by showing how our previous protocols can be adapted to work with ECW and AfP instead of standard encryption and authentication. We then show how the optimizations in the DDH based constructions via the SCRAPE trick carry over to our anonymous setting if we instantiate our ECW and AfP schemes from similar assumptions. The protocols we construct in this section work in the YOSO model supporting up to $t < n/2$ corrupted parties and can be used as efficient building blocks for the protocols of [BGG+20, GHK+21].

In the previous sections, we have constructed both a PVSS scheme (Section 7.3.2) and a PVSS re-sharing scheme (Section 7.4.1) based on $\mathbb{Z}_p$-linear encryption schemes (as defined in Section 5.8.3). Despite being efficient, these constructions are not fit for the YOSO model because they require the dealer to know the public keys of the parties who will receive shares, consequently revealing their identities. In order to solve this issue, we show that these protocols can also be instantiated with the ECW scheme of Section 7.2 even though they were designed to be instantiated with a $\mathbb{Z}_p$-linear encryption scheme. The core idea is that our ECW preserves all the properties of the underlying $\mathbb{Z}_p$-linear encryption scheme while adding the ability to encrypt towards a role rather than towards a party who owns a public key.

7.5.1 Constructing HEPVSS with ECW

We modify HEPVSS to use our ECW scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ for lottery predicate $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ from Section 7.2 instead of a $\mathbb{Z}_p$-linear encryption scheme. Departing from the HEPVSS algorithms described in Fig. 7.3, we make the following modifications:

- Communication: All messages are posted to the underlying blockchain ledger used by the ECW scheme $\mathcal{E}$.
- $\mathsf{HEPVSS.Setup}(1^\lambda)$: Besides the original setup parameters, we assume that n distinct role identifiers $\mathsf{R}_1, \ldots, \mathsf{R}_n$ are available and that an underlying blockchain protocol Γ is executed.
- $\mathsf{HEPVSS.KeyGen}(pp, id)$: Instead of publishing pk_i, each party P_i provides pk_i as input to the mixnet assumed as setup for $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,i})$ and associated ECW scheme $\mathcal{E}$. The mixnet output $\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}$ is assumed to be available on the underlying blockchain and accessible as

  $$(\{(j, \mathsf{pk}_{\mathsf{Anon},j})\}_{j\in[n]}, \eta) \leftarrow \mathsf{param}(\mathbf{B}, \mathsf{sl}).$$

  Party P_i sets $\mathsf{sk}_{L,i} \leftarrow (\mathsf{pk}_{\mathcal{E},i}, \mathsf{sk}_{\mathcal{E},i})$.
- $\mathsf{HEPVSS.Dist}(pp, \{\mathsf{pk}_i : i \in [n]\}, S)$: Instead of computing $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(A_i, \rho_i)$, the dealer computes $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, A_i)$ using randomness ρ_i. Notice that this is equivalent to computing $C_i \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},j}}(A_i, \rho_i)$ for a j such that

$\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,j}) = 1$. Hence, $\mathsf{Pf}_{\mathsf{Sh}}$ can still be computed via the same procedure. The dealer publishes

$$(\{C_i : i \in [n]\}, \{\mathsf{pk}_{\mathsf{Anon},j} : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}).$$

Notice that the public key $\mathsf{pk}_{\mathsf{Anon},j}$ used to generate each C_i is publicly known due to the structure of the lottery scheme.

- HEPVSS.Verify$(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$: No modification is needed, since $(\{C_i : i \in [n]\}, \{\mathsf{pk}_{\mathsf{Anon},j} : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$ has the same structure as in the original protocol.

- HEPVSS.DecShare$(pp, \mathsf{pk}_j, \mathsf{sk}_{L,j}, C_i)$: Party P_j checks that its lottery witness $\mathsf{sk}_{L,j}$ is such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,j}) = 1$ and, if yes, computes $A_i \leftarrow \mathsf{Dec}(\tilde{\mathbf{B}}, C_i, \mathsf{sk}_{L,j})$. Proof $\mathsf{Pf}_{\mathsf{Dec}}$ is generated as in the original protocol. Notice that this procedure is also equivalent to generating an AfP $\mathsf{Pf}_{\mathsf{Dec}} \leftarrow \mathsf{AfP.Sign}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,j}, A_i)$.

- HEPVSS.VerifyDec$(pp, \mathsf{pk}_i, A_i, C_i, \mathsf{Pf}_{\mathsf{Dec}i})$: Proof $\mathsf{Pf}_{\mathsf{Dec}}$ is checked as in the original protocol. Notice that this procedure is also equivalent to generating an AfP $\{0,1\} \leftarrow \mathsf{AfP.Ver}(\tilde{\mathbf{B}}, \mathsf{sl}, \mathsf{R}_i, \mathsf{Pf}_{\mathsf{Dec}}, A_i)$.

- HEPVSS.Rec$(pp, \{A_i : i \in \mathcal{T}\}$: No modification is needed.

Due to the properties of the ECW scheme and the underlying lottery scheme, shares are encrypted towards parties randomly chosen to perform each role R_i whose identity remains unknown during the share distribution and verification phases. In case a reconstruction happens, parties executing each role reveal themselves by proving correctness of decrypted shares, which constitutes an AfP since it involved proving knowledge of $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,j}) = 1$.

7.5.2 Constructing Resharing for HEPVSS with ECW

In the context of resharing, the parties selected to execute roles $\mathsf{R}_1, \ldots, \mathsf{R}_n$ in slot sl_r wish to publicly verifiable reshare the secret whose shares they received towards roles $\mathsf{R}'_1, \ldots, \mathsf{R}'_{n'}$ in a future slot $\mathsf{sl}_{\mathsf{r}+1}$. In practice, this means that the resharing information will be received by a new randomly selected set of anonymous parties performing these roles in the future. Once again we explore the fact that our ECW inherits the properties of the underlying $\mathbb{Z}_p$-linear encryption scheme to modify the resharing protocol of Fig. 7.7 to work with ECW.

We show how to modify the description of Fig. 7.7 to obtain an ECW based resharing protocol:

- **Participants:** Parties executing roles $\mathsf{R}_1, \ldots, \mathsf{R}_n$ in slot sl_r and parties executing roles $\mathsf{R}'_1, \ldots, \mathsf{R}'_{n'}$ in slot $\mathsf{sl}_{\mathsf{r}+1}$.

- **Input:** Public (*i.e.* published in the underlying blockchain) ECW ciphertexts $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_\mathsf{r}, \mathsf{R}_i, A_{tr,i})$ such that $A_{\mathsf{r},i} = f_\mathsf{r}(\alpha_i) \cdot G$ for some polynomial f_r of degree $\leq t_\mathsf{r}$.

- **Output:** ECW ciphertexts $C_i \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_{r+1}, \mathsf{R}'_i, A_{r,i})$ published in the underlying blockchain such that $A_{r+1,k} = f_{r+1}(\beta_k) \cdot G$ for some polynomial f_{r+1} of degree $\leq t_{r+1}$ and $f_{r+1}(\beta_0) = f_r(\alpha_0)$.

- **Protocol:**

  - *Encryption/Decryption:* When decrypting ciphertexts using key sk_i for $i \in [n_r]$, ECW decrypt using $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_r, \mathsf{R}_i, \mathsf{sk}_{L,j}) = 1$. When encrypting a message under public key pk_j for $j \in [n_{r+1}]$, ECW encrypt towards role R'_j in slot sl_{r+1} using randomness $\rho_{r+1,,j}$: $C_j \leftarrow \mathsf{Enc}(\mathbf{B}, \mathsf{sl}_{r+1}, \mathsf{R}_{r+1,j}, A)$. Notice that this is equivalent to computing $C_j \leftarrow \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_{\mathsf{Anon},r+1,j}}(A, \rho_{r+1,j})$ for a j such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}_{r+1}, \mathsf{R}_j, \mathsf{sk}_{L,j}) = 1$.

  - *Proof* $\mathsf{HEPVSS.Reshare.Verify}(pp, \mathsf{pk}_{r,i}, \{\mathsf{pk}_{r+1,j}\}_{j\in[n_{r+1}]}, C_{r,i}, \{C_{i\to j}\}_{j\in[n_{r+1}]}, \pi_i)$*:* Notice that the structure of the ECW ciphertexts is compatible with this proof, so that it can be generated as in the original protocol. Analogously, this proof can also be verified as in the original protocol. Moreover, notice that it also acts as an AfP for ciphertexts $\{C_{i\to j} : j \in [n_{r+1}]\}$ on behalf of role R_i of slot sl_r, since it requires knowledge of a $\mathsf{sk}_{L,j}$ such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}_i, \mathsf{sk}_{L,j}) = 1$.

As in the PVSS with ECW protocol, due to the properties of the ECW scheme and the underlying lottery scheme, resharing information is encrypted towards parties randomly chosen to perform each role $\mathsf{R}_{r+1,j}$ whose identity remains unknown until they act (*e.g.* by reconstructing the secret).

7.5.3 Efficient DDH-based Instantiation via DHPVSS

The most efficient instantiations of our techniques are obtained when using a variant of the El Gamal encryption scheme together with the SCRAPE share validity check. In order to enjoy the efficiency improvement, we show that our ECW is also compatible with these optimizations.

- **Setup and Lottery Predicate:** We use the same setup, *i.e.* we assume the parties have access to an ideal mixnet and input their public keys E_i so that the output of a tuple $\{(j, E_{\mathsf{Anon},j}) : j \in [n]\}$ which is written on the blockchain and accessible to every party through param function. The lottery predicate works the same way, having parties check whether $E_{\mathsf{Anon},k} = E_i$ for $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{R}||\eta)$ in order to determine if they have been selected for role R in slot sl. Moreover, every party publishes on the underlying blockchain a public key $\mathsf{pk}_{D,i}$ for which they know the corresponding secret key $\mathsf{sk}_{D,i}$, which they will use when encrypting.

- **Encryption:** As in our original ECW a party P_i encrypting m towards role R in slot sl starts by running $\mathsf{param}(\mathbf{B}, \mathsf{sl})$ to obtain $(\{(l, E_{\mathsf{Anon},l})\}_{l\in[n]}, \eta)$ and determine $E_{\mathsf{Anon},k}$ such that $k \leftarrow \mathcal{H}(\mathsf{sl}||\mathsf{R}||\eta)$. P_i publishes ciphertext $C_{i,k} \leftarrow m + \mathsf{sk}_{D,i} \cdot E_k$ revealing indices i, k. Notice that this ciphertext has exactly the same structure as the ciphertexts used in DHPVSS.

- **Decryption:** To decrypt a ciphertext $C_{i,k}$ for role R in slot sl, P_j checks that its $\mathsf{sk}_{L,j}$ is such that $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,j}) = 1$. If yes, it obtains the sender's public key $\mathsf{pk}_{D,i}$ from the blockchain and computes $m \leftarrow C_{i,k} - \mathsf{sk}_j \cdot \mathsf{pk}_{D,i}$. Notice that a proof of correct decryption can be done exactly as in DHPVSS and that such a proof constitutes an AfP of m on behalf of role R in slot sl, since it requires proving knowledge of $\mathsf{sk}_{L,j}$ s.t. $\mathsf{lottery}(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \mathsf{sk}_{L,j}) = 1$.

Using this slight modification of our ECW, we can instantiate DHPVSS (Figs. 7.4 and 7.5) and its resharing protocol (Fig. 7.7). The ciphertexts output by ECW have the same structure as those used in DHPVSS, so the efficient proofs of encrypted (re)share validity can be performed exactly in the same way.

Privacy and Resharing: Notice, however, that since the dealer's identity must be known when decrypting ciphertexts, using these optimized techniques for resharing will be problematic, since it requires linking a party P_i to its key $\mathsf{sk}_{D,i}$ and revealing its identity. In order to solve this issue, we can resort to a similar setup used for the regular keys E_i, *i.e.* we can allow parties access to an ideal mixnet that is used to create a shuffled set of keys $\{(j, \mathsf{sk}_{D,\mathsf{Anon},j}) : j \in [n]\}$. Now a sender can include the index to its key $\mathsf{sk}_{D,\mathsf{Anon},j}$ in the ciphertext in order to allow for decryption. As it is the case with our simple AfP technique, this would require setting up multiple such vectors, which can potentially be solved by techniques similar to those we describe in Section 7.9. We leave a concrete description of such a construction for future works.

7.6 Proofs for ECW

In this section we list the proofs related to theorems stated in Section 7.2. We re-state the theorems for convenience.

Theorem 7.4 (IND-CPA ECW). *Let $\mathcal{E}$ be an IND-CPA secure $\mathbb{Z}_p$-linearly homomorphic encryption scheme. The construction in Fig. 7.1 with lottery predicate as in Section 7.2.1 is an IND-CPA secure ECW (as in Definition 6.2).*

Proof (Sketch). An adversary with a noticable advantage in $\mathsf{Game}^{\mathsf{IND\text{-}CPA}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{E}}$ described in Definition 6.2 can distinguish between ECW encryptions of two different messages without winning the lottery for that specific sl and R. This adversary can, in turn, distinguish between corresponding encryptions from the underlying $\mathbb{Z}_p$-linearly homomorphic encryption scheme $\mathcal{E}$, which contradicts IND-CPA security of $\mathcal{E}$. Thus, the protocol in Fig. 7.1 yields an IND-CPA secure ECW.
IND-CCA security for the ECW scheme can be obtained by using standard transformations ([FO99, Sah99]) as argued in [CDK+22].

Theorem 7.5 (EUF-CMA AfP). *Let $\mathcal{E}$ be an IND-CPA secure and $\mathbb{Z}_p$-linearly homomorphic encryption scheme and let* SoK *be a simulatable and extractable SoK scheme. The con-*

struction in Fig. 7.2 with lottery predicate as in Section 7.2.1 is EUF-CMA AfP as defined in Definition 6.5.

Proof (Sketch). We argue that an adversary who forges a signature (AfP tag) on a message m is able to construct a valid SoK on a message without knowing the witness. More precisely, assume that the adversary can make the verifier output $b = 1$ on input $(\mathbf{B}, \mathsf{sl}, \mathsf{R}, \sigma, m)$ while not having won the lottery for parameters $(\mathbf{B}, \mathsf{sl}, \mathsf{R})$. The underlying SoK σ_{SoK} must be a convincing SoK on m such that $\mathsf{SoK.verify}((\mathbf{B}, \mathsf{sl}, \mathsf{R}), \sigma_{\mathsf{SoK}}, m) = 1$. Thus, the adversary has successfully created a SoK signature where the verification algorithm accepts but without the adversary knowing a witness. This breaks existential unforgeability of the SoK scheme contradicting our assumption.[1]

Theorem 7.6 (AfP Privacy)**.** *Assume $\mathcal{E}$,* lottery *and* SoK *scheme as in 7.5. The construction in Fig. 7.2 has AfP privacy as in Definition 6.6.*

Proof (Sketch). We construct a simulator $\mathcal{S}$ for the game $\mathsf{Game}^{\text{ID-PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ as follows. When $\mathcal{S}$ gets a request for given tuple $(\mathbf{B}, \mathsf{sl}, \mathsf{R}, m)$ it forwards the request to the simulator for the SoK scheme. The SoK simulator can forge a signature and, in particular, it can simulate an SoK on m without knowing the lottery winning secret key. Then, $\mathcal{S}$ obtains the response of the SoK simulator forwards it to the adversary. We claim that any adversary who can successfully distinguish between interacting with the simulator $\mathcal{S}$ and the oracle $\mathcal{O}_{\mathsf{AfP}}$ in $\mathsf{Game}^{\text{ID-PRIV}}_{\Gamma,\mathcal{A},\mathcal{Z},\mathcal{U},\mathcal{E}}$ breaks the simulatability of the SoK scheme.

7.7 Other Security Proofs

7.7.1 Correctness and Security of HEPVSS

Lemma 7.1 (Correctness of HEPVSS)**.** *If $\mathcal{E}$ is correct then construction* HEPVSS *satisfies* correctness.

Proof. Recall correctness means in this case that if keys $(\mathsf{pk}_i, \mathsf{sk}_i)$ have been created honestly with HEPVSS.KeyGen, a secret $S \in \mathbb{G}$ has been distributed according to HEPVSS.Dist resulting in encrypted shares C_i and a proof $\mathsf{Pf}_{\mathsf{Sh}}$, these shares C_i have been correctly decrypted resulting in A'_i and proofs $\mathsf{Pf}_{\mathsf{Dec}\,i}$, and a secret S' is reconstructed from these A'_i, then the verification of $\mathsf{Pf}_{\mathsf{Sh}}$ and the $\mathsf{Pf}_{\mathsf{Dec}\,i}$ accept and $S' = S$.

Note that by definition of GShamir, the dealer creates $A_i = S + m(\alpha_i) \cdot G$ for a polynomial $m(X)$ of degree at most t with $m(\alpha_0) = 0$ and $C_i = \mathcal{E}.\mathsf{Enc}(A_i, \rho_i)$. $\mathsf{Pf}_{\mathsf{Sh}}$ asserts precisely this. Clearly f as defined in the proof is a linear map and $\Pi_{\mathsf{NI-Pre}}$ is correct, so $\mathsf{Pf}_{\mathsf{Sh}}$ will be accepted. By correctness of $\mathcal{E}$, the decrypted A'_i will equal A_i. By correctness of

[1] In fact, forging a signature in the EUF-CMA game of SoK reduces to either breaking the corresponding simulatability or the extractability of the SoK scheme (see [CL06])

$\mathcal{E}$.ProveDec the proofs $\mathsf{Pf}_{\mathsf{Dec}\,i}$ are accepted. By definition of the reconstruction in GShamir outputs S when applied to any subset $A_i, i \in \mathcal{T}$ where $\mathcal{T}$ is of size $t+1$. □

Theorem 7.7 (IND-1 Secrecy). *If $\mathcal{E}$ is IND-CPA then construction* HEPVSS *for a PVSS satisfies* indistinguishability of secrets

Proof. Let $\mathcal{A}$ be an adversary that can win the IND-1 Secrecy Game for HEPVSS with non-negligible advantage ϵ. Note that as a dealer secret key is not needed to perform HEPVSS.Dist, we do not need to consider the DIST oracle. We construct an adversary $\mathcal{B}$ that uses to $\mathcal{A}$ to break $(n-t)$-multi-key IND-CPA security, which is equivalent to IND-CPA security (see Section 5.8).

Firstly, $\mathcal{B}$ passes the keys pk_i, $i \in [n-t]$ to $\mathcal{A}$ and observes its constructed pk_i, $i \in [n-t+1, n]$. Then $\mathcal{B}$ chooses random polynomials $m^{(0)}$, and $m^{(1)}$ of degree at most t under the restriction that $m^{(0)}(\alpha_i) = m^{(1)}(\alpha_i)$ for i in $[n-t+1, n]$. [2]

$\mathcal{B}$ sets $S_0 = m^{(0)}(\alpha_0)\cdot G$, $S_1 = m^{(1)}(\alpha_0)\cdot G$, $A_i^{(0)} = m(\alpha_i)\cdot G$, $A_i^{(1)} = m(\alpha_i')\cdot G$ for $i \in [n]$ and sends message vectors $\mathbf{m}^{(j)} = (A_1^{(j)}, A_2^{(j)}, \ldots, A_{n-t}^{(j)})$ for $j \in \{0,1\}$ to the IND-CPA challenger, and receives $(C_1, \ldots, C_{n-t})$ in return where $C_i = \mathsf{Enc}_{\mathsf{pk}_i}(A_i^{(b)})$.

Now $\mathcal{B}$ computes $C_i = \mathsf{Enc}(m^{(0)}(\alpha_i)\cdot G)$ for $i \in [n-t+1, n]$ and note that for these values of i, $m^{(0)}(\alpha_i) = m^{(1)}(\alpha_i)$ so $C_i = \mathsf{Enc}(m^{(1)}(\alpha_i)\cdot G)$ too. Finally given C_i, $i \in [n]$, $\mathcal{B}$ constructs a simulated proof $\mathsf{Pf}_{\mathsf{Sh}}{}^*$. Now $\mathcal{B}$ sends $C_i, i = 1, \ldots, n$, and $\mathsf{Pf}_{\mathsf{Sh}}{}^*$ to $\mathcal{A}$ as well as the candidate secret S_0. $\mathcal{B}$ then outputs the same guess as $\mathcal{A}$.

It is clear that $\mathcal{A}$ receives from $\mathcal{B}$ encrypted shares of S_0 (if the challenger's bit is $b=0$) or S_1 (if $b=1$) distributed identically as in the protocol: indeed the ciphertexts $C_1, \ldots, C_{n-t}$ are the encryptions of either the set $\{A_i^{(0)}\}_{i\in[n-t]}$ or $\{A_i^{(1)}\}_{i\in[n-t]}$ of the first $n-t$ shares constructed by $\mathcal{B}$ for the secrets, and the last t ciphertexts created by $\mathcal{B}$ are encryptions of $A_i^{(0)} = A_i^{(1)}$, $i \in [n-t+1, n]$. Finally $\mathsf{Pf}_{\mathsf{Sh}}{}^*$ is computationally indistinguishable from a real proof of correct sharing by the zero knowledge property of the proof. Therefore the guessing advantage of $\mathcal{B}$ for the multi-key IND-CPA game is the same as that of $\mathcal{A}$. □

Lemma 7.2 (Verifiability of HEPVSS). *Construction* HEPVSS *for a publicly verifiable secret sharing scheme satisfies* verifiability.

Proof. Verifiability of Key Generation. Our construction clearly satisfies verifiability of key generation because public keys simply consist of one group element, and so it is easy to verify public keys are correctly formed.

Verifiability of Distribution. Our construction satisfies verifiability of distribution because if HEPVSS.Verify$(pp, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1$ then $\mathsf{Pf}_{\mathsf{Sh}}$ is a valid proof of witness $w = (S, m(X), \rho_1, \ldots, \rho_n)$ such that $m(\alpha_0) = 0$, for all $i \in [n]$ $C_i = \mathcal{E}.\mathsf{Enc}_{\mathsf{pk}_i}(S + m(\alpha_i)\cdot G, \rho_i)$

[2] Which can be done by choosing first $m^{(0)}(X)$ and then taking $m^{(1)}(X) = m^{(0)}(X) + \gamma \cdot \prod_{i=n-t+1}^{n}(X-\alpha_i)$ for uniformly random γ.

and m has degree $\leq t$. Therefore, clearly HEPVSS.Dist on input $pp, \{\mathsf{pk}_i : i \in [n]\}, S$ and with randomness $m(X), \rho_1, \ldots, \rho_n$ will output $\{C_i : i \in [n]\}$.

Verifiability of Decryption. Our construction clearly satisfies verifiability of decryption because if

$$\text{HEPVSS.VerifyDec}(pp, \mathsf{pk}, A, C, \mathsf{Pf}_{\mathsf{Dec}}) = 1$$

then $\mathsf{Pf}_{\mathsf{Dec}}$ is a valid proof of witness sk such that $A = \mathsf{Dec}_{\mathsf{sk}}(C)$ and sk is the secret key corresponding to pk. Therefore, $\text{DecShare}(pp, \mathsf{pk}, \mathsf{sk}, C) = (A, \cdot)$ for any randomness input to this algorithm. □

7.7.2 Correctness and Security of DHPVSS

Lemma 7.3 (Correctness of DHPVSS)**.** *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* correctness*.*

Proof. Consider a set of encrypted shares $\{C_i : i \in [n]\}$ and a proof $\mathsf{Pf}_{\mathsf{Sh}}$ output by DHPVSS.Dist with respect to parameters $pp = (\mathbb{G}, G, p, \{\alpha_i, v_i : i \in [n]\})$ output by DHPVSS.Setup, a secret $S \in \mathbb{G}$, a public and secret key $(\mathsf{pk}_D, \mathsf{sk}_D)$ generated by DHPVSS.DKeyGen, and a set of public keys $\{\mathsf{pk}_i : i \in [n]\} = \{(E_i = \mathsf{sk}_i \cdot G, \Omega_i) : i \in [n]\}$ generated by DHPVSS.KeyGen with respect to $\{id_i : i \in [n]\}$.

Clearly for all $i \in [n]$, $\text{VerifyKey}(pp, id_i, \mathsf{pk}_i) = 1$, as because of the correctness of the proofs of discrete logarithms, Ω_i will be valid.

For all $i \in [n]$, $\text{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}_i, \mathsf{sk}_i, C_i)$ outputs $A_i = C_i - \mathsf{sk}_i \cdot \mathsf{pk}_D$ and $\mathsf{Pf}_{\mathsf{Dec}\,i}$. Then, by definition of correctness, DHPVSS is correct if

$$\text{DHPVSS.Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1,$$

for all $i \in [n]$

$$\text{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk}_i, C_i, A_i, \mathsf{Pf}_{\mathsf{Dec}\,i}) = 1,$$

and finally DHPVSS.Rec outputs the secret S.

Consider the proof $\mathsf{Pf}_{\mathsf{Sh}}$ where

$$V = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i, \quad U = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i,$$

where $f^* = H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$ and $\forall i \in [n]$

$$v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1}.$$

By assumption, the proofs of discrete logarithm equality are correct. As $C_i = \mathsf{sk}_D \cdot E_i + A_i$ where $A_i = S + m(\alpha_i) \cdot G$ for polynomial m of degree $\leq t$ such that $m(\alpha_0) = 0$, then

$$V = \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i = \mathsf{sk}_D \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i + \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot (S + m(\alpha_i) \cdot G)$$

$$= \mathsf{sk}_D \cdot U + \sum_{i=1}^{n} v_i f^*(\alpha_i)(S + m(\alpha_i) \cdot G).$$

As m is a polynomial of degree t such that $m(\alpha_0) = 0$ and due to Theorem 5.1, $V = \mathsf{sk}_D \cdot U$. As $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$, then the proof $\mathsf{Pf}_{\mathsf{Sh}}$ will be valid. Therefore, algorithm DHPVSS.Verify returns 1.

We consider now $\mathsf{Pf}_{\mathsf{Dec}i}$. By assumption, the proofs of discrete logarithm equality are correct. Because $C_i - A_i = \mathsf{sk}_i \cdot \mathsf{pk}_D$ and $E_i = \mathsf{sk}_i \cdot G$, then the proof $\mathsf{Pf}_{\mathsf{Dec}i}$ will be valid. Therefore, $\forall i \in [n]$ algorithm DHPVSS.VerifyDec returns 1.

Finally, clearly DHPVSS.Rec will output GShamir.Rec$(pp, \{A_i : i \in \mathcal{T}\})$ which in turn equals S since A_i are all correct. □

Lemma 7.4 (IND-1 Secrecy of DHPVSS)**.** *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* indistinguishability of secrets *if the DDH assumption holds.*

Proof. Suppose there is an adversary $\mathcal{A}$ such that

$$\Pr\left[\mathsf{Game}^{\text{ind-secrecy},0}_{\mathcal{A},\mathsf{PVSS}}(\lambda) = 1\right] - \Pr\left[\mathsf{Game}^{\text{ind-secrecy},1}_{\mathcal{A},\mathsf{PVSS}}(\lambda) = 1\right] = \epsilon,$$

where ϵ is non-negligible, then we can construct $\mathcal{B}$ that distinguishes DDH tuples with non-negligible probability. We give the detailed description of $\mathcal{B}$ in Algorithm 17, and then explain how $\mathcal{B}$ works.

We now explain why, when a DDH tuple is input to $\mathcal{B}$, the view of $\mathcal{A}$, when $j = 1$ and $\tilde{b} = 1$, is as in the real experiment when $b = 1$ and the view of $\mathcal{A}$, when $j = n - t$ and $\tilde{b} = 0$, is as in the real experiment when $b = 0$. Note that $\tilde{b}$ and j are the values randomly chosen by $\mathcal{B}$.

G and pk_D are set to be X_1, X_2 respectively and so are distributed correctly. All E_i for $i \in [n - t]$ are chosen identically to the experiment,except for E_j which is X_3 a random element of $\mathbb{G}$ and so distributed correctly. We can simulate the proof Ω_j due to the zero knowledge property of the proof of discrete logarithms. When computing the encrypted shares, although the secret key sk_D is not known to $\mathcal{B}$, we can instead use sk_i such that $\mathsf{sk}_i \cdot G = E_i$ for all $i \in [n]$. We have that $\mathsf{sk}_D \cdot E_i = \mathsf{sk}_i \cdot \mathsf{pk}_D$. In the case of corrupted parties, although we do not know these sk_i values, we can extract them from the proofs of knowledge Ω_i. In the case of the jth honest party, although again we do not know sk_j, as a DDH tuple is input, then $X_4 = \mathsf{sk}_D \cdot E_j$ where $\mathsf{sk}_D \cdot G = \mathsf{pk}_D$. The proof $\mathsf{Pf}_{\mathsf{Sh}}$ can be simulated without knowledge of sk_D, due to the zero knowledge property. The oracle DIST can be simulated without knowledge of sk_D in the same way. The sk_i values that were extracted from the public keys E_i can be used to generate $\{C'_i : i \in [n]\}$. The proof $\mathsf{Pf}_{\mathsf{Sh}}$ can again be simulated.

For all corrupted parties, it does not matter whether the polynomial f or f' is used to generate their encrypted share, because they have the same outputs on input α_i where $i \in [n - t + 1, n]$. When $j = 1$, and $\tilde{b} = 1$, the polynomial f' is used to generate all of

Algorithm 17: $\mathcal{B}$

procedure DIST$((\mathcal{U}, S'))$

  if $\mathcal{U} \not\subseteq [n+1, k]$ or $|\mathcal{U}| \neq n$ **then return** $\perp$
  end if
  Let $\{(\alpha_i, v_i) : i \in \mathcal{U}\}$ be the set $\{(\alpha_i, v_i) : i \in [n]\}$
  $(\{A'_i\}_{i\in[n]}, m'(X)) \leftarrow \mathsf{GShamir.Share}((\mathbb{G}, G, p, t, n, \{\alpha_i : i \in \mathcal{U}\}), S')$
  $\forall i \in \mathcal{U} \quad C'_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + A'_i$
  $f^* \leftarrow H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C'_i) : i \in \mathcal{U}\})$
  $V \leftarrow \sum_{i\in\mathcal{U}} v_i f^*(\alpha_i) \cdot C'_i$
  $U \leftarrow \sum_{i\in\mathcal{U}} v_i f^*(\alpha_i) \cdot E_i$
  Simulate proof $\mathsf{Pf}_{\mathsf{Sh}}$ for G, pk_D, U, V
  return $(\{C'_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$
end procedure

procedure $\mathcal{B}(\mathbb{G}, p, X_1, X_2, X_3, X_4)$
  $\tilde{b} \leftarrow\$ \{0, 1\}, j \leftarrow\$ [1, n-t]$
  $G \leftarrow X_1, x_0, x_1 \leftarrow\$ \mathbb{Z}_p, S_0 \leftarrow x_0 \cdot G, S_1 \leftarrow x_1 \cdot G$
  Choose pairwise distinct $\alpha_1 \in \mathbb{Z}_p, \cdots \alpha_n \in \mathbb{Z}_p$
  $\forall i \in [n] \quad v_i \leftarrow \prod_{j\in[n]\setminus\{i\}} (\alpha_i - \alpha_j)^{-1}$
  Randomly sample degree t polynomials $f, f' \in \mathbb{Z}_p[X]$ with $f(0) = x_0$, $f'(0) = x_1$, and $f(\alpha_i) = f'(\alpha_i)$ for $i \in [n-t+1, n]$
  $pp \leftarrow (\mathbb{G}, G, p, \{(\alpha_i, v_i) : i \in [n]\}); \mathsf{pk}_D \leftarrow X_2$
  $E_j \leftarrow X_3$; simulate the proof Ω_j for G, E_j, j; $\mathsf{pk}_j \leftarrow (E_j, \Omega_j)$
  $\forall i \in [n-t]\setminus\{j\} \quad \mathsf{sk}_i \leftarrow\$ \mathbb{Z}_p, E_i \leftarrow \mathsf{sk}_i \cdot G, \Omega_i \leftarrow \mathsf{DL}(\mathsf{sk}_i; G, E_i, i); \mathsf{pk}_i \leftarrow (E_i, \Omega_i)$
  $((\{\mathsf{pk}_i = (E_i, \Omega_i) : i \in [n-t+1, k]\}) \leftarrow \mathcal{A}(pp, \mathsf{pk}_D, \{\mathsf{pk}_i : i \in [n-t]\})$
  $\forall i \in [n-t+1, k]$ extract sk_i from Ω_i
  $\forall i \in [1, j-1] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f(\alpha_i) \cdot G$
  $\forall i \in [j+1, n-t] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f'(\alpha_i) \cdot G$
  $\forall i \in [n-t+1, n] \quad C_i \leftarrow \mathsf{sk}_i \cdot \mathsf{pk}_D + f(\alpha_i) \cdot G$

  if $\tilde{b} = 0$ **then** $C_j \leftarrow X_4 + f(\alpha_j) \cdot G$
  end if

  if $\tilde{b} = 1$ **then** $C_j \leftarrow X_4 + f'(\alpha_j) \cdot G$
  end if
  $f^* \leftarrow H(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$
  $V \leftarrow \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot C_i, U \leftarrow \sum_{i=1}^{n} v_i f^*(\alpha_i) \cdot E_i$
  Simulate proof $\mathsf{Pf}_{\mathsf{Sh}}$ for G, pk_D, U, V
  $b' \leftarrow \mathcal{A}^{\mathsf{DIST}}(S_0, \{C_i : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}})$

  if $b' = \tilde{b}$ **then return** 1
  else return 0
  end if
end procedure

the encrypted shares for the honest parties. Therefore, the adversary is input S_0 and a correctly distributed sharing for S_1 and so the view is identically distributed to when $b = 1$. When $j = n-t$ and $\tilde{b} = 0$, the polynomial f is used to generate all of the encrypted shares for the honest parties. The adversary is input S_0 and a correctly distributed sharing for S_0 and so the view to $\mathcal{A}$ is identically distributed to when $b = 0$.

Let $W_{j,d}$ be respectively the event that $\mathcal{A}$ outputs d when j is chosen at the beginning and a DDH tuple is input to $\mathcal{B}$. Where ϵ is the advantage of $\mathcal{A}$ defined above which is non–negligible, we have that

$$|\Pr\left[W_{n-t,1}|\tilde{b}=0\right] - \Pr\left[W_{1,1}|\tilde{b}=1\right]| = \epsilon.$$

Note that for $j^* = 1, \ldots, n-t-1$, the view of the adversary when $j = j^*+1$ and $\tilde{b} = 1$ and the view of the adversary when $j = j^*$ and $\tilde{b} = 0$ is identically distributed so $\Pr\left[W_{j^*+1,1}|\tilde{b}=1\right] = \Pr\left[W_{j^*,1}|\tilde{b}=0)\right]$. Then

$$|\Pr\left[W_{n-t,1}|\tilde{b}=0\right] - \Pr\left[W_{1,1}|\tilde{b}=1\right]| =$$

$$\left|\sum_{j=1}^{n-t}\left(\Pr\left[W_{j,1}|\tilde{b}=0\right] - \Pr\left[W_{j,1}|\tilde{b}=1\right]\right)\right|.$$

When a DDH tuple is input to $\mathcal{B}$, the probability $\mathcal{B}$ outputs 1 is

$$\frac{\sum_{j=1}^{n-t} 1/2\Pr\left[W_{j,1}|\tilde{b}=1\right] + 1/2(1-\Pr\left[W_{j,1}|\tilde{b}=0\right])}{n-t}$$

$$= 1/2 + \frac{\sum_{j=1}^{n-t}\Pr\left[W_{j,1}|\tilde{b}=1\right] - \Pr\left[W_{j,1}|\tilde{b}=0\right])}{2(n-t)}.$$

Now consider the probability that $\mathcal{B}$ outputs 1 when a random tuple was input to $\mathcal{B}$. Because X_4 is now a uniform and independent variable, all inputs to $\mathcal{B}$ are independant of $\tilde{b}$. Therefore, $\mathcal{B}$ outputs 1 with probability $1/2$.

As $|\frac{\sum_{j=1}^{n-t}\Pr\left[W_{j,1}|\tilde{b}=1\right]-\Pr\left[W_{j,1}|\tilde{b}=0\right])}{2(n-t)}| = \frac{\epsilon}{2(n-t)}$, which is non-negligible, then $\mathcal{B}$ has a non-negligible advantage in distinguishing DDH tuples. □

Lemma 7.5 (Verifiability). *Our construction* DHPVSS *for a publicly verifiable secret sharing scheme satisfies* verifiability *if the hash function H is a random oracle.*

Proof. Verifiability of Key Generation. Our construction clearly satisfies verifiability of key generation because if $\mathsf{VerifyKey}(pp, id, \mathsf{pk} = (E, \Omega)) = 1$ then Ω is a valid proof of knowledge of the discrete logarithm for E. Therefore, sk such that $E = \mathsf{sk} \cdot G$ can be extracted from Ω.

Verifiability of Distribution. Our construction satisfies verifiability of distribution because if

$$\mathsf{Verify}(pp, \mathsf{pk}_D, \{(\mathsf{pk}_i = (E_i, \Omega_i), C_i) : i \in [n]\}, \mathsf{Pf}_{\mathsf{Sh}}) = 1$$

then $\mathsf{Pf}_{\mathsf{Sh}}$ is a valid proof for the fact that the discrete logarithm of pk_D with respect to G, is the same as that of V with respect to U, where

$$V = \sum_{i=1}^{n} v_i m^*(\alpha_i) \cdot C_i, \quad U = \sum_{i=1}^{n} v_i m^*(\alpha_i) \cdot E_i$$

and

$$m^* = \mathcal{H}(\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\}), \quad v_i = \prod_{j \in [n] \setminus \{i\}} (\alpha_i - \alpha_j)^{-1} \; \forall i \in [n].$$

Therefore, sk_D such that $\mathsf{pk}_D = \mathsf{sk}_D \cdot G$ and $V = \mathsf{sk}_D \cdot U$ can be extracted from $\mathsf{Pf}_{\mathsf{Sh}}$. As $V = \mathsf{sk}_D \cdot U$, then

$$\sum_{i=1}^{n} v_i \cdot m^*(\alpha_i) \cdot (C_i - \mathsf{sk}_D \cdot E_i) = 0.$$

Let Φ denote the event

$$(C_1 - \mathsf{sk}_D \cdot E_1, \cdots, C_n - \mathsf{sk}_D \cdot E_n) \neq (f(\alpha_1) \cdot G, \cdots, f(\alpha_n) \cdot G)$$

for every polynomial f of degree $\leq t$. Say r queries were made to the random oracle by the adversary. For event Φ to have occurred, some $\mathsf{pk}_D, \{(\mathsf{pk}_i, C_i) : i \in [n]\})$ was submitted to the random oracle and some polynomial m^* of degree $\leq n - t - 1$ was returned such that $\sum_{i=1}^{n} v_i m^*(\alpha_i) \cdot (C_i - \mathsf{sk}_D \cdot E_i) = 0$. As $E_1, \cdots E_n$ are included in the input to the hash function and sk_D is defined by the input to the hash function, the probability of this is at most r/p, due to Theorem 5.1. Now assume Φ did not happen. Then there has to be a polymonial f satisfying the conditions above. Then, letting $S = f(\alpha_0) \cdot G$, and $(\{C_i : i \in [n]\}, \cdot) = \mathsf{DHPVSS.Dist}(pp, \mathsf{pk}_D, \mathsf{sk}_D, \{\mathsf{pk}_i : i \in [n]\}, S)$ where the randomness r is the one that makes GShamir.Share select polynomial $m(X) = f(X) - S$ [3]. Therefore, clearly a correctly formed sk_D, S and randomness for Dist exist.

Verifiability of Decryption. Our construction clearly satisfies verifiability of decryption because if $\mathsf{VerifyDec}(pp, \mathsf{pk}_D, \mathsf{pk} = (E, \Omega), C, A', \mathsf{Pf}_{\mathsf{Dec}}) = 1$ then $\mathsf{Pf}_{\mathsf{Dec}}$ is a valid proof of knowledge of discrete logarithm equality for $G, E, \mathsf{pk}_D, C - A'$. Therefore, sk such that $E = \mathsf{sk} \cdot G$ and $C - A' = \mathsf{sk} \cdot \mathsf{pk}_D$ can be extracted from $\mathsf{Pf}_{\mathsf{Dec}}$. Therefore $A' = C - \mathsf{sk} \cdot \mathsf{pk}_D$, and so $\mathsf{DecShare}(pp, \mathsf{pk}_D, \mathsf{pk}, \mathsf{sk}, C) = (A', \cdot)$ for any randomness input to this algorithm. □

7.8 Communication Complexity of PVSS

First note that any $\Pi_{\mathsf{NI-Pre}}$ communicates an element in $\mathcal{W}$ and one in $\mathbb{Z}_p$.

[3] recall that GShamir.Share constructs the shares as $A_i = S + m(\alpha_i)G$ for m of degree $\leq t$ with $m(\alpha_0) = 0$; the above selection of m satisfies the conditions and yields $A_i = f(\alpha_i)$

Communication Complexity of HEPVSS**.** The communication of the algorithm HEPVSS.Dist consists of n ciphertexts in $\mathfrak{C}$ (the encryptions if the shares) and a proof $\mathsf{Pf}_{\mathsf{Sh}}$, which is a $\Pi_{\mathsf{NI-Pre}}$ proof where $\mathcal{W} = \mathbb{G} \times \mathbb{Z}_p[X]_{\leq t} \times \mathfrak{R}^n$. When El Gamal encryption is used as $\mathcal{E}$, since $\mathfrak{R} = \mathbb{Z}_p$, $\mathfrak{C} = \mathbb{G}^2$ this amounts to a total of $(n + t + 2)$ elements of $\mathbb{Z}_p$ and $2n + 1$ in $\mathbb{G}$ which is roughly [4] equivalent to a total of $(3n + t + 3) \log p$ bits.

On the other hand HEPVSS.DecShare communicates a decrypted message in $\mathbb{G}$ and the proof $\mathsf{Pf}_{\mathsf{Dec}}$ where $\mathcal{W} = \mathcal{SK}$. In the case where we use El Gamal, the latter is 1 element in $\mathbb{G}$[5] and a challenge in $\mathbb{Z}_p$. Hence the communication is roughly $3 \log p$ bits.

Communication Complexity of DHPVSS**.** DHPVSS.Dist has smaller ciphertexts (1 group element each) and a smaller proof $\mathsf{Pf}_{\mathsf{Sh}}$ consisting only of 2 elements in $\mathbb{Z}_p$. Hence the communication is in total roughly $(n + 2) \log p$ bits, which is $3 \sim 3.5\times$ less than HEPVSS.Dist (depending on t).

We remark that this is quite close to the minimum possible, at least if one uses an information-theoretical secret sharing scheme, where the public communication is made through encryption of the shares, as we do. Indeed, well known bounds imply that, in this case, the total joint size of the shares must be n times the secret, therefore $n \log p$ bits in our situation.

The communication of DHPVSS.DecShare is as in HEPVSS.DecShare, hence it communicates $3 \log p$ bits.

Comparison with SCRAPE and ALBATROSS. In SCRAPE and ALBATROSS, the encrypted shares of a secret $S = m(\alpha_0)G$ are given by $C_i = m(\alpha_i)\mathsf{pk}_i$ (where again $\mathsf{pk}_i = \mathsf{sk}_i G$). SCRAPE requires the dealer to commit to $m(\alpha_i)$ in a common base H by publishing $M_i = m(\alpha_i)H$ (n additional group elements), so that the SCRAPE trick can be used on the M_i's. Moreover the dealer needs to post non-interactive $\mathsf{DLEQ}(m(\alpha_i), \mathsf{pk}_i, C_i, H, M_i)$ for all i, which amounts to $n+1$ new $\mathbb{Z}_p$-elements. In total this means $(3n+1) \log p$ bits for the whole distribution. Instead ALBATROSS uses a standard homomorphic preimage proof of knowledge of the $m(X)$ underlying C_i. That is the dealer posts $\Pi_{\mathsf{NI-Pre}}(m(X), \{C_i\}_{i=1,n}, f)$ with $f(m(X)) = m(\alpha_i) \cdot \mathsf{pk}_i$. This requires $t + 2$ $\mathbb{Z}_p$-elements, and so the communication complexity of the distribution phase is of $(n + t + 2) \log p$ bits. Therefore, our DHPVSS scheme is the most communication efficient of all these alternatives.

Communication Complexity of Resharing. Resharing a secret among a committee of n_{r+1} parties requires, per party that is resharing their share, $(3n_{r+1} + t_{r+1} + 3) \log p$ bits, i.e. the same communication as to execute HEPVSS.Dist among the same set of parties. This means that we need a total communication of $(t_r + 1)(3n_{r+1} + t_{r+1} + 3) \log p$ bits in order for $\mathcal{C}_r$ to reshare a secret to $\mathcal{C}_{r+1}$.

[4] In practice, describing an element of an elliptic-curve group of order p requires slightly more information

[5] While it is true that, in order to force linearity of decryption, we have artificially set $\mathsf{sk}^* = (1, \mathsf{sk})$, and hence the keys are technically in $\mathbb{G}^2$, it is very easy to see that one only needs to send information related to the second coordinate.

The same happens with DHPVSS: the communication complexity per party who is re-sharing is $(n_{r+1} + 2) \log p$ bits, which is the same as for distributing a share in the first place. This means $\mathcal{C}_r$ needs to communicate in total $t_r(n_{r+1} + 2) \log p$ bits to reshare a secret to $\mathcal{C}_{r+1}$.

7.9 Zero Knowledge Proofs of Membership to an Anonymous Committee

When encryption towards a committee $\mathcal{R}$ is used as part of a protocol, identities $\mathcal{P}_i$ such that $\psi(i) \in \mathcal{R}$ will typically need to act upon having received an encrypted message. This will reveal the fact that they are a receiver.

In this section we present strategies that allow $\mathcal{P}_i$ to prove that it belongs to the receiver set, $\psi(i) \in \mathcal{R}$, without revealing anything else about $\psi(i)$.

At first, it could appear that having $\mathcal{P}_i$ prove knowledge of a message received by $\mathcal{R}$ is enough, but note this is not the case when there are collusions between corrupted parties in $\mathcal{R}$ and others outside. In general, we want to avoid that a set of t colluding parties of which only $t' < t$ belong to $\mathcal{R}$ can claim that $t' + 1$ or more of them are in $\mathcal{R}$.

We present two solutions for the problem above. The first solution (Section 7.9.1) is generic but less efficient: each party in $\mathcal{R}$ signs a message using a linkable ring signature [LWW04]. Ring signatures [RST01] guarantee that the signer belongs to a given set of parties without revealing their identity within that set. Linkability ensures that, despite this anonymity, two signatures using the same key can be linked. This means colluding parties cannot use the same secret key to claim that both belong to $\mathcal{R}$, when only one of them does.

However, linkable rings signatures become larger as the size of the committee grows. In Section 7.9.2, we present an optimized solution where we leverage the fact that, in our situation, there is already a sender broadcasting ciphertexts, and we can use this party to send auxiliary information that allows to reduce the amount of communication by each receiver to be constant-size (while the information sent by the sender is still linear in the size of the receiver committee). Our solution is based on a linkable version of Camenisch-Lysyanskaya signatures.

7.9.1 Generic Proofs of Membership based on Linkable Ring Signatures

Ring signatures, also called sometimes Spontaneous Anonymous Group signatures, are signature schemes in which each member of a universe of parties has a secret key, and can use that key to sign a message on behalf of any subset of that universe to which it belongs, in such a way that the signature does not reveal which of the parties in that subset has signed.

Ring signatures can be constructed as non-interactive zero knowledge proofs of knowl-

edge of a secret key corresponding to a set of public keys (which is in turn an OR statement), via a Fiat-Shamir transformation where the message is included as an argument to the random oracle. In fact it is this proof of knowledge what we really need in our problem, but we present the solution in terms of ring signatures because the notion of linkability is commonly used in this context. A linkable ring signature is one that guarantees that if two signatures (even of different messages) for the same set of users are produced using the same secret key, this fact is detected, even though the identity of the signer is kept anonymous.

Definition 7.6 (Linkable Ring Signature)**.** *A Linkable Ring Signature scheme for a set* $[n]$ *is given by the following tuple of algorithms:*

- $\mathsf{KeyGen}(n, 1^\lambda)$*: Outputs n key pairs* $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]}$.
- $\mathsf{LinkSig}(\mathsf{sk}_i, m, \mathcal{R})$*: Takes a secret key, a message* m*, and a set* $\mathcal{R} \subseteq [n]$*, outputs a signature* σ.
- $\mathsf{LinkVer}(\{\mathsf{pk}_i\}_{i\in\mathcal{R}}, m, \sigma, \mathcal{R})$*: Takes a set* $\mathcal{R} \subseteq [n]$*, a set of associated public keys* pk_i*,* $i \in \mathcal{R}$*, a message* m *and a signature* σ *and outputs* $\mathtt{accept}$ *or* $\mathtt{reject}$.
- $\mathsf{Link}((m, \sigma, \mathcal{R}), (m', \sigma', \mathcal{R}'))$*: Takes two tuples consisting of a message, a signature and a subset of* $[n]$ *and outputs a bit* b *(meant to represent whether these two signatures have been created with the same secret key).*

In addition, these algorithm must satisfy the following properties: for all messages m, m_0, m_1*, all sets* $\mathcal{R}, \mathcal{R}_0, \mathcal{R}_1 \subseteq [n]$*, all* $(\mathsf{pk}_i, \mathsf{sk}_i)_{i\in[n]}$ *output by* $\mathsf{KeyGen}(n, 1^\lambda)$ *and any* $\mathsf{sk}, \mathsf{sk}^{(0)}, \mathsf{sk}^{(1)} \in (\mathsf{sk}_i)_{i\in[n]}$

$$\Pr[\mathsf{LinkVer}(\{\mathsf{pk}_i\}_{i\in\mathcal{R}}, m, \sigma, \mathcal{R}) = \mathtt{accept}) |\ \sigma = \mathsf{LinkSig}(\mathsf{sk}_i, m, \mathcal{R}) \wedge i \in \mathcal{R}] = 1$$

$$\Pr[\mathsf{Link}((m_0, \sigma_0, \mathcal{R}_0), (m_1, \sigma_1, \mathcal{R}_1)) = 1 |\ \sigma_b = \mathsf{LinkSig}(\mathsf{sk}, m_b, \mathcal{R}_b), b \in \{0, 1\}] = 1$$

$$\Pr[\mathsf{Link}((m_0, \sigma_0, \mathcal{R}_0), (m_1, \sigma_1, \mathcal{R}_1)) = 1 |\ \sigma_b = \mathsf{LinkSig}(\mathsf{sk}^{(b)}, m_b, \mathcal{R}_b), b \in \{0, 1\} \wedge \mathsf{sk}^{(0)} \neq \mathsf{sk}^{(1)}] = 0$$

The first equation ensures that a signature σ of a message m is always accepted by a verifier that takes as additional input a set $\mathcal{R}$ and the public keys corresponding to that set, if the signature has been created with a secret key belonging to $\mathcal{R}$. The second and third equations guarantee that two signatures of two possibly different messages (and with respect to possibly different sets) will be linked if and only if they have been created with the same key.

Typically several security properties are required from linkable ring signatures, which we describe informally. These are based on the model in [BDH+19].

- **Linkability** This requirement ensures that signatures from the same secret key will always be linked. In the security game, the adversary must output k public keys for corrupted parties, and $k + 1$ valid signatures, each on a message and a ring. They

win if all rings are subsets of the set of the k corrupted public keys, and none of the signatures are linked. The requirement is that the adversary wins with negligible probability

- **Linkable Anonymity** While linkable ring signatures are publicly linkable, a signature still should not be able to be traced to the signer's public key. In the game, the adversary is given access to an oracle to create honest users and receive their public keys. The adversary returns two honest users (their challenged users), as well as a set of the adversary's own corrupted public keys. They are then given access to an oracle, where they can submit a challenged user, a message and a ring that must contain the public keys of both challenged users. The challenger returns a signature signed with the secret key of one of the users and the adversary must guess the signer correctly to win. The adversary must have negligible advantage in guessing correctly.

- **Non–Frameability** This requirement ensures that an adversary cannot frame an honest user by forging a signature which links to this user's signature. In the game we give the adversary access to oracles to create honest users, obtain their signatures and corrupt them. The adversary must output a valid signature that was not output by the signing oracle. They then must output another valid signature that was output by the signing oracle for an honest user that has not been corrupted. For the adversary to win, the two signatures must be valid and linked. This should happen with negligible probability.

Note that linkability implies the usual existential unforgeability security property, in the sense that, if the adversary knows no secret key $\mathsf{sk}_j, j \in \mathcal{R}$ (i.e. $|\mathcal{C} \cap \mathcal{R}| = 0$) then the adversary cannot create a valid signature for $\mathcal{R}$.

Linkable ring signatures almost automatically gives a solution to our problem. Each party includes a public key $\mathsf{pk}_{\mathsf{LinkSig},i}$ for a linkable signature in the public key to be shuffled. To prove membership to $\mathcal{R}$, $\mathcal{P}_i$ signs a message with $\mathsf{pk}_{\mathsf{LinkSig},i}$ and publishes the message and signature. This signature can be verified by any public verifier. The security properties of the linkable signature guarantee both that the proof only reveals membership to $\mathcal{R}$ but nothing else, and that if two identities use the same secret key to claim membership to $\mathcal{R}$, this is detected by any public verifier.

One (easily fixable) caveat is that the properties above do not prevent replay attacks, where an adversary attempts to copy an honest party's signature and claim it as theirs, or at least invalidate the honest party's signature. We fix this by including the public identity of the signer as a part of the message signed. We describe the construction in Fig. 7.9.

Proof of membership to an anonymous committee

Set-up:
A list of anonymized public keys $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$ where $\mathsf{pk}_{\mathsf{Anon},j}$ contains a public key $\mathsf{pk}_{\mathsf{LinkSig},j}$ for a linkable signature scheme.
For the sake of notation simplicity, let $\mathsf{pk}_j := \mathsf{pk}_{\mathsf{LinkSig},j}$, $\mathsf{sk}_j := \mathsf{sk}_{\mathsf{LinkSig},j}$ in this Figure. Moreover, given $\mathcal{R} \subseteq [n]$, let $\mathcal{PK}_{\mathcal{R}} = \{\mathsf{pk}_j : j \in \mathcal{R}\}$.

Proof:
Input: A subset $\mathcal{R} \subseteq [n]$, a session identifier ssid, the key sk_j known to $\mathcal{P}_i$.

1. Set $m = H(\mathcal{P}_i||\mathsf{ssid})$ and output $\sigma = \mathsf{LinkSig}(\mathsf{pk}_j, m, \mathcal{R})$.

Verification:
Input: Public verifier has input $\mathcal{R}, \mathcal{PK}_{\mathcal{R}}$, ssid, and a list of pairs $(\mathcal{P}_i, \sigma_i)$, $i \in I$ for a subset $I \subseteq \mathcal{P}$.

1. Let $m_i = H(\mathcal{P}_i||\mathsf{ssid})$ for $i \in I$
2. Compute the set $I' \subseteq I$ of all i in I such that $\mathsf{LinkVer}(\mathcal{PK}_{\mathcal{R}}, m_i, \sigma_i, \mathcal{R}) = \texttt{accept}$.
3. For all $i_1, i_2 \in I'$, $i_1 \neq i_2$, if $\mathsf{Link}((m_{i_1}, \sigma_{i_1}, \mathcal{R}), (m_{i_2}, \sigma_{i_2}, \mathcal{R})) = 1$ then remove i_1, i_2 from I'. Continue until there are no such index pairs.
4. Output the remaining set I' as the set of accepted membership claims.

Figure 7.9: Proof of membership verification

We require that if a set I of users have all generated proofs of membership to an anonymous committee honestly, then verification will pass. This is clearly true, due to the correctness of linkable ring signatures. We require three security requirements for our proof of membership to an anonymous committee:

- **Unforgeability** This requirement ensures that proofs of membership from the same party in an anonymous committee can be linked. In the game, the adversary has corrupted t parties in a anonymous committee $\mathcal{R}$ of size R. They can see proofs of membership from honest parties and must output $t+1$ proofs of memberships on the corrupted identities, i.e. for ID_i such that i has been corrupted. They win if these proofs of memberships pass verification.

  Clearly this is true for our construction in Fig. 7.9, due to the linkability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the unforgeability game for proofs of membership to an anonymous committee, we can win in the linkability game for linkable ring signatures. The adversary in our unforgeability game provides us with t public keys corresponding to corrupted users and we generate ourselves $R - t$ secret/public keypairs corresponding to honest users. We can then honestly generate $R-t$ proofs of membership on behalf of honest users to provide to the adversary. They return $t+1$ proofs of membership on behalf of corrupted users. In the linkability game we output all R public keys of all honest and corrupted members of the anonymous committee, and all $R+1$ proofs of memberships on behalf of corrupted and honest members of the anonymous committee. As all proofs of membership

pass verification, we have output $R + 1$ valid ring signatures that are all unlinked. Therefore, we have broken the linkability of linkable ring signatures.

- **Anonymity** Although a proof of membership of an anonymous committee reveals that the prover is a member of $\mathcal{R}$, we need to ensure that it does not reveal which member of $\mathcal{R}$. In the game, the adversary chooses two honest users in $\mathcal{R}$ and has corrupted all other users. They then receive a proof of membership on behalf of one of the honest users, and must guess which user correctly to win.

  Clearly this is true for our construction in Fig. 7.9, due to the linkable anonymity requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the anonymity game for proofs of membership to an anonymous committee, we can win in the linkable anonymity game for linkable ring signatures. We first of all create two honest users in the linkable anonymity game. We can set the public keys of the two honest users, chosen by the adversary in the anonymity game for proofs of membership, to be these two public keys. We then submit to the challenge oracle one of these honest users, along with a ring containing all public keys in the anonymous committee and a message set to be $H(ID, \mathsf{ssid})$. We return the resulting ring signature as our proof of membership in the anonymity game, and finally return the resulting bit b output by the adversary. If the adversary wins in the anonymity game, we clearly win in the linkable anonymity game, which is a contradiction.

- **Non–Frameability** This requirement ensures that an adversary cannot frame an honest user by forging a proof of membership which links to this user's proof of membership, therefore implying unfairly that they cheated. In the game, we give the adversary access to the public keys of honest users, and oracles to obtain their proofs of membership. The adversary must output a proof of membership that was not output by the oracle. They then must output another proof of membership that was output by the signing oracle for an honest user. For the adversary to win, the two signatures must not pass verification together, but should pass verification individually.

  Clearly this is true for our construction in Fig. 7.9, due to the non–frameability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the non–frameability game for proofs of membership to an anonymous committee, we can win in the non–frameability game for linkable ring signatures. We will provide the adversary in the non–frameability game for proofs of membership with the public keys of honest users, using the corresponding oracle in the non–frameability game for linkable ring signatures. When the adversary in the non–frameability game for proofs of membership attempts to obtain the proofs of memberships for honest users, we will use the signing oracle in the non–frameability game for linkable ring signatures. The adversary in the non–frameability game for proofs of membership will output two proofs of membership that individually pass verification, but fail together: one output from the signing oracle for an honest user and one that was not output by the signing oracle. We can then output these two proofs of membership in the linkable ring signature game. They will both be valid and linked signatures, so we will win in the non-frameability game for linkable ring signatures.

7.9.2 Efficient Instantiation using Camenisch-Lysyanskaya Signatures

In this section, we propose a solution where the size of a membership proof is constant (independent from the size of $\mathcal{R}$). For this we leverage the fact that the sender can send auxiliary information together with the ciphertexts. Our strategy is based on a "linkable version" of a signature scheme by Camenisch-Lysyanskaya.

We focus on one version of the Camenisch-Lysyanskaya signatures which has been used for anonymous credentials and where we want to construct a signature of a group element $sG \in \mathbb{G}$. A crucial feature of this proof is that it can be divided in two parts: the first part uses the signing key and does not require knowledge of s and outputs σ; meanwhile, the second part is a proof of knowledge of s and does not require to know the secret signing key.

This means that the two parts of the proof can be carried out by two different parties. Moreover the signature has a second important property: if the owner of the signature key has carried out the first part of the signing for different $s_i G$, with outputs σ_i, then the second part of the signature (the proof of knowledge) does not reveal which σ_i is being completed.

Our strategy is then the following. The sender carries out the first part of the CL signature of each of the public keys of the parties in $\mathcal{R}$, thereby creating messages σ_i. Now, because of what we mentioned above, any receiver can prove the knowledge of the discrete logarithm of one of these secret keys, without revealing which.

As before, this has the problem that, if a party ID_i in $\mathcal{R}$ is colluding with other parties outside the set, then they could all use the secret key known by ID_i and claim to be in $\mathcal{R}$. In order to prevent that we turn the signature into a linkable one by including another generator H in the common reference string, and having each receiver publish $I_j = \mathsf{sk}_{\mathsf{Anon},j} H$. We extend the proof of knowledge of $\mathsf{sk}_{\mathsf{Anon},j}$ into one that ensures $\mathsf{sk}_{\mathsf{Anon},j}$ is the same as the discrete log of I_j in base H. Since I_j is deterministically computed from H and $\mathsf{sk}_{\mathsf{Anon},j}$, a verifier can easily check if two parties have claimed the same key.

Camenisch-Lysyanskaya Signatures

The precise signature we will use is the one called Signature A in [CL04], but with the difference that while that paper assumed a type I bilinear pairing (which would not allow for using the DDH assumption), we will replace it by a Type III bilinear pairing as has been done in other works such as [CDL16, GNQT20].

We recall this signature scheme: Let $\mathbb{G}_1, \mathbb{G}_2$ (with additive notation) and $\mathbb{G}_T$ (with multiplicative notation) be groups of prime order p. Let $\mathbb{G}_1$ be generated by G_1 and $\mathbb{G}_2$ be generated by G_2. The signing secret key is of the form $\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2$ and the public key $\mathsf{pk}_{\mathsf{CL}} = (X, Y) = (xG_2, yG_2)$ in $\mathbb{G}_2^2$.

The signature scheme can be used to either sign messages $m \in \mathbb{Z}_p$ or $M = mG_1 \in \mathbb{G}_1$. We are interested in the latter case. As mentioned above, this case can be separated in

two algorithms, where $\mathsf{CL.Sig}^1$ uses M and $\mathsf{sk}_{\mathsf{CL}}$ but does not require knowledge of m, and $\mathsf{CL.Sig}^2$ is applied to the output of $\mathsf{CL.Sig}^1$ and requires knowledge of m, but not of the secret key. These protocols are defined in Fig. 7.10.

Camenisch-Lysyanskaya signature

Setup: Groups $(\mathbb{G}_1, +), (\mathbb{G}_2, +), (\mathbb{G}_T, \cdot)$ of order p with generators G_1, G_2 for $\mathbb{G}_1, \mathbb{G}_2$ respectively, bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. A random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$.
Parties and keys: A sender has a CL keypair $(\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2, \mathsf{pk}_{\mathsf{CL}} = (X, Y))$ where $X = xG_2$, $Y = yG_2$.

$\mathsf{CL.Sig}^1_{\mathsf{sk}_{\mathsf{CL}}}(M)$:

1: **parse** $(x, y) \leftarrow \mathsf{sk}_{\mathsf{CL}}$
2: $a \leftarrow\$ \mathbb{Z}_p$, $A \leftarrow aG_1 \in \mathbb{G}_1$
3: $B \leftarrow yA$, $C \leftarrow xA + axyM$
4: **return** $\sigma^1 \leftarrow (A, B, C)$

▷ Note that if we call $M = mG$, then $C = xA + axyM = (x + mxy)A$.

$\mathsf{CL.Sig}^2(m, \sigma^1)$:

1: Parse σ^1 as (A, B, C)
2: $r, r' \leftarrow\$ \mathbb{Z}_p^*$
3: $\tilde{A} \leftarrow r'A$, $\tilde{B} \leftarrow r'B$, $\hat{C} \leftarrow rr'C$
4: $z_A \leftarrow e(\tilde{A}, X)$, $z_B \leftarrow e(\tilde{B}, X)$, $z_C \leftarrow e(\hat{C}, G_2)$
5: $\rho \leftarrow r^{-1}$
6: $\mathcal{W} \to \mathbb{Z}_p^2$, $\mathcal{X} \leftarrow \mathbb{G}_T$, $pp_\pi \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
7: $\pi \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}((\rho, m); pp_\pi, z_A, f_{(z_B, z_C)})$
8: where $f_{(z_B,z_C)}(\rho, m) = z_B^{-m} z_C^{\rho}$.

▷ This proves knowledge of ρ and m such that $z_B^{-m} z_C^{\rho} = z_A$.

9: **return** $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, \pi)$

$\mathsf{Ver}^2(\mathsf{PK}, \sigma^2)$:

1: Parse $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, \pi)$
2: Compute z_A, z_B, z_C as in $\mathsf{CL.Sig}^2$ above.
3: **return** accept iff $e(\tilde{A}, Y) = e(\tilde{B}, G_2)$ and $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(pp_\pi, z_A, f_{(z_B,z_C)})$ accepts.

Figure 7.10: Camenisch-Lysyanskaya signature

A crucial point is that the verification step depends only on the output of $\mathsf{CL.Sig}^2$. Moreover, given $(M_1, \sigma_1^1), \ldots, (M_n, \sigma_n^1)$ where $M_i = m_i G_1$ and $\sigma_i^1 = \mathsf{CL.Sig}^1_{(x,y)}(M_i)$, the signature $\mathsf{CL.Sig}^2(m_i, \sigma_i^1)$ gives no information about i

This means that, once $\mathsf{CL.Sig}^1$ has been carried out on the messages M_i, the second step of the signature can be seen as a ring signature scheme of sorts: if we interpret (m_i, M_i) as a secret key/public key pair belonging to the i-th party in a given set of parties, as it will be our case, then by executing $\mathsf{CL.Sig}^2$ on the output of $\mathsf{CL.Sig}^1_{(x,y)}(M_i)$ the i-th party is creating a signature (for an "empty" message) that guarantees this party belongs to the set, without revealing their identitity.

Adding linkability

To ensure linkability in the scenario we just described, namely that any verifier can detect when $\mathsf{CL.Sig}^2$ has been applied twice on the same input, we do the following:

First, as part of the setup we fix H, a generator of group $\mathbb{G}_1$, as part of the set up. Then $\mathsf{CL.LinkSig}^2(m, \sigma^1)$ works as follows:

Algorithm 18: $\mathsf{CL.LinkSig}^2(m, \sigma^1)$

$I \leftarrow mH$
Compute $(\tilde{A}, \tilde{B}, \hat{C}, \pi')$ as in $\mathsf{CL.Sig}^2(m, \sigma^1)$ except now
  $\pi' \leftarrow \Pi_{\mathsf{NI-Pre}}((\rho, m); pp_\pi, (z_A, I), f'_{(z_B, z_C, H)})$,
  where $f'_{(z_B, z_C, H)}(\rho, m) := (z_B^{-m} z_C^{\rho}, mH)$.
  and $pp_\pi = (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}', \mathcal{H})$ where $\mathcal{X}' = \mathbb{G}_T \times \mathbb{G}_1$
return $\sigma^2 \leftarrow (\tilde{A}, \tilde{B}, \hat{C}, I, \pi')$

Now I depends deterministically on m and public information, and therefore a verifier can detect if the same m is used twice, as it will yield the same I.

Final instantiation

Our final instantiation, described formally in Fig. 7.11 is as follows: The sender will encrypt the message with the El Gamal encryption scheme under the anonymous public keys in $\mathcal{R}$ and include a proof of correctness of encryption. Moreover, the sender will compute $\sigma_j^1 = \mathsf{CL.Sig}^1_{\mathsf{sk_{CL}}}(\mathsf{pk}_{\mathsf{Anon},j})$ for $j \in \mathcal{R}$, where $\mathsf{sk_{CL}}$ is the secret key for the sender. Finally, we observe that in the description of CL signatures above there is no guarantee that σ_j^1 has been computed correctly until Ver^2 is executed, so we need the sender to additionally prove that σ_j^1 is indeed computed correctly from $\mathsf{CL.Sig}^1(\mathsf{pk}_{\mathsf{Anon},j})$. To claim membership to $\mathcal{R}$, and therefore ownership of some $\mathsf{sk}_{\mathsf{Anon},j}$, a party can then compute $\sigma^2 = \mathsf{CL.Sig}^2(\mathsf{sk}_{\mathsf{Anon},j}, \sigma_j^1)$. As in the generic construction, to avoid replay attacks we add the public identity of the prover in the argument of the Fiat-Shamir random oracle for the proof of knowledge π.

Encryption to an anonymous committee via CL signatures

Setup: Groups $(\mathbb{G}_1, +), (\mathbb{G}_2, +), (\mathbb{G}_T, \cdot)$ of order p. Generators G_1, H for $\mathbb{G}_1$, generator G_2 for $\mathbb{G}_2$, bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

Parties and keys: A sender has a CL keypair $(\mathsf{sk}_{\mathsf{CL}} = (x, y) \in \mathbb{Z}_p^2, \mathsf{pk}_{\mathsf{CL}} = (X, Y))$ where $X = xG_2$, $Y = yG_2$.
In addition, there is a set $\mathcal{P}$ of potential receivers. In the setup phase, every party chooses a keypair $(\mathsf{sk}, \mathsf{pk})$ where $\mathsf{sk} \in \mathbb{Z}_p$, $\mathsf{pk} = \mathsf{sk}G_1 \in \mathbb{G}_1$ and then inputs it to a mix-net, resulting in a public list $\{(j, \mathsf{pk}_{\mathsf{Anon},j}) : j \in [n]\}$.

EncAMC.Enc$(M, \mathsf{sk}_{\mathsf{CL}}; \mathcal{R}, (\mathsf{pk}_{\mathsf{Anon},j})_{j\in\mathcal{R}})$ **where** $M \in \mathbb{G}_1$**,** $\mathcal{R} \subseteq [n]$:

1: $\forall j \in \mathcal{R}$, $r_j \leftarrow\$ \mathbb{Z}_p$
2: $\forall j \in \mathcal{R}$, $c_j \leftarrow \mathcal{E}.\mathsf{Enc}_{pk_{\mathsf{Anon},j}}(M; r_j)$
3: $\pi_{\mathsf{EC}} \leftarrow \mathcal{E}.\mathsf{ProveEnc}(M, (r_j)_{j\in\mathcal{R}}; (c_j)_{j\in\mathcal{R}})$,
4: $\forall j \in \mathcal{R}$, $\sigma_j^1 \leftarrow \mathsf{CL.Sig}^1_{\mathsf{sk}_{\mathsf{CL}}}(\mathsf{pk}_{\mathsf{Anon},j})$ with randomness $a_j \in \mathbb{Z}_p$
5: $\pi_{\mathsf{CLSC}} \leftarrow \mathsf{CLSC.Prove}(x, y, (a_j)_{j\in\mathcal{R}}; X, Y, (\mathsf{pk}_{\mathsf{Anon},j})_{j\in\mathcal{R}}, (\sigma_j^1)_{j\in\mathcal{R}})$,
6: as in Fig. 7.12 below (this proves that σ_j^1 are correct CLSC signatures)
7: **return** $((c_j, \sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}}, \pi_{\mathsf{CLSC}})$

EncAMC.Ver$((c_j, \sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}}, \pi_{\mathsf{CLSC}})$:

1: **return** accept iff
2: both $\mathcal{E}.\mathsf{VerifyEnc}((c_j)_{j\in\mathcal{R}}, \pi_{\mathsf{EC}})$ and $\mathsf{CLSC.Verify}((\sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{CLSC}})$ accept.

EncAMC.Claim$(\mathsf{sk}_{\mathsf{Anon},j}; (\sigma_j^1)_{j\in\mathcal{R}})$:

1: **return** $\sigma^2 \leftarrow \mathsf{CL.LinkSig}^2(\mathsf{sk}_{\mathsf{Anon},j}, \sigma_j^1)$

EncAMC.ClaimVer$(\{(\sigma_i^2)_{i\in\mathcal{I}}\}, \mathsf{pk}_{\mathsf{CL}})$:

1: Receive as input a set $(\sigma_i^2)_{i\in\mathcal{I}}$ of verification claims
2: For all $i \in \mathcal{I}$, parse $\sigma_i^2 = (\tilde{A}_i, \tilde{B}_i, \hat{C}_i, I_i, \pi_i')$.
3: **for** each $I \in \mathbb{G}_T$ such that there are more than one $i \in \mathcal{I}$ with $I_i = I$ **do**
4: Let $\mathcal{I}_I$ the set of such i.
5: **if** there is exactly one i in $\mathcal{I}_I$ such that $\mathsf{Ver}^2(\mathsf{pk}_{\mathsf{CL}}, \sigma_i^2)$ accepts **then**
6: **Accept** this claim and reject all other claims from parties in $\mathcal{I}_I$
7: **else**
8: **Reject** all membership claims from parties in $\mathcal{I}_I$
9: **end if**
10: **end for**
11: **for** each I such that there is one $i \in \mathcal{I}$ with $I_i = I$ **do**
12: **Accept** the claim if and only if $\mathsf{Ver}^2(\mathsf{pk}_{\mathsf{CL}}, \sigma_i^2)$ accepts
13: **end for**

Figure 7.11: Encryption to a committee with anonymous membership claim

Proof of Camenisch-Lysyanskaya signature correctness

Proof for relation

$$\begin{aligned} R_{\mathsf{CLSC}} = \quad & \{((x, y, (a_j)_{j\in\mathcal{R}}); (X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma_j^1)_{j\in\mathcal{R}})) : \\ & \sigma_j^1 = (a_j \cdot G_1, a_j \cdot y \cdot G_1, a_j \cdot x \cdot G_1 + a_j \cdot x \cdot y \cdot \mathsf{pk}_{\mathsf{Anon},j}), \\ & X = x \cdot G_1, \\ & Y = y \cdot G_1\} \end{aligned}$$

$\mathsf{CLSC.Prove}(x, y, (a_j)_{j\in\mathcal{R}}; X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma_j^1)_{j\in\mathcal{R}})$:

Let $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ random oracle.

1: **for** j in $\mathcal{R}$ **do**
2: **parse** $\sigma_j^1 \leftarrow (\sigma_{j1}^1, \sigma_{j2}^1, \sigma_{j3}^1)$.
3: $b_j \leftarrow a_j y$, $c_j \leftarrow a_j x$, $d_j \leftarrow a_j xy$. ▷ Introducing these new variables "linearizes" the problem
4: **end for**
5: $\mathcal{W} \leftarrow \mathbb{Z}_p^{4|\mathcal{R}|}$, $\mathcal{X} \leftarrow \mathbb{G}_1^{6|\mathcal{R}|}$, $pp \leftarrow (\mathbb{Z}_p, \mathcal{W}, \mathcal{X}, \mathcal{H})$
6: **return** $\pi_{\mathsf{CLSC}} \leftarrow \Pi_{\mathsf{NI-Pre}}.\mathsf{Prove}(w; pp, x, f)$ where
7: $w = (a_j, b_j, c_j, d_j)_{j\in\mathcal{R}}$
8: $x = (\sigma_{j1}^1, \sigma_{j2}^1, \sigma_{j3}^1, O, O, O)_{j\in\mathcal{R}}$
9: $f(w) := (a_j G_1, b_j G_1, c_j G_1 + d_j \mathsf{pk}_j, a_j Y_D - b_j G_1, a_j X - c_j G_1, c_j Y - d_j G_1)_{j\in\mathcal{R}}$
10: (Note that, for each $j \in \mathcal{R}$, the first 3 conditions in f check the target statement using the introduced variables, while the three last ensure these variables are correctly defined)

$\mathsf{CLSC.Verify}(X, Y, (\mathsf{pk}_j)_{j\in\mathcal{R}}, (\sigma_j^1)_{j\in\mathcal{R}}, \pi_{\mathsf{CLSC}})$:

1: **return** $\Pi_{\mathsf{NI-Pre}}.\mathsf{Verify}(x, f, \pi_{\mathsf{CLSC}})$ where x, f are defined from the σ_j^1 as above.

Figure 7.12: Proof of Camenisch-Lysyanskaya Signature Correctness CLSC

The correctness of the EncAMC scheme is satisifed, due to the correctness of the Camenisch-Lysyanskaya signatures. Clearly the prooofs π_{EC} and π_{CLSC} guarantee that the sender has behaved honestly. We again require three security requirements for our proof of membership to an anonymous committee as defined previously:

- **Unforgeability** Clearly this is true for our construction in Fig. 7.11, due to the LRSW assumption [LRSW99], which ensures the security of Camenisch-Lysyanskaya signatures. We now provide a proof sketch. For an adversary to have output $k+1$ proofs of memberships that pass verification and that were not honestly generated, after having corrupted t of the public keys, they must have returned $t+1$ signatures that are valid according to Ver^2, containing elements $I_1, \cdots I_{t+1}$ with for all $(i, j) \in [t+1]$ $I_i \neq I_j$. Say $\exists i \in [t+1]$ such that $I_i = \mathsf{sk}H$, where sk is the secret key of an honest user. Then we can build an adversary that can break the discrete logarithm, by extracting sk due to the proof of knowledge property. Say $\exists i \in [k+1]$ such that $I_i = \mathsf{sk}H$, where sk is not the secret key of any user (corrupt or honest). Then we can build an adversary that can break the unforgeability of CL signatures, because the adversary has forged a signature on a new message $\mathsf{sk}G$. Now it is not possible for all $I_1, \cdots, I_{t+1}$ to be distinct, as there are only t corrupted users, and so we have a contradiction.

- **Anonymity** Clearly this is true for our construction in Fig. 7.11, due to the DDH as-

sumption. We now provide a proof sketch. We show that given an adversary that can win in the anonymity game for proofs of membership to an anonymous committee, we can distinguish DDH tuples. We are input $X_1, X_2, X_3, X_4 \in \mathbb{G}_1^4$. In setup we set $G = X_1$, $H = X_2$. We choose bit $b \leftarrow\$ \{0,1\}$ and set the public key of the bth honest user to be X_3. We then generate a proof of membership as follows. We set $I = X_4$, and choose $\tilde{A}$, $\tilde{B}$, $\tilde{C}$ as normal based on the signature $\sigma^1 = (A, B, C)$ of the bth honest user. We then simulate the attached proof, which is possible to the zero knowledge property. If the adversary guesses correctly, we output 1, and otherwise we output 0. If a DDH tuple is input, then the inputs to the adversary in the proof of membership game are distributed correctly and we output 1 with the same probability that the adversary is successful. If a DDH tuple is not input, then the inputs to the adversary are independent of b, and we output 1 with probability 1/2.

- **Non–Frameability** Clearly this is true for our construction in Fig. 7.11, due to the non–frameability requirement for linkable ring signatures. We now provide a proof sketch. We show that given an adversary that can win in the non–frameability game for proofs of membership to an anonymous committee, we can break the discrete logarithm assumption. We are input $X_1, X_2 \in \mathbb{G}_1^2$. In setup we set $G = X_1$, $H = aG$, where $a \leftarrow\$ \mathbb{Z}_p$. When the adversary in the non–frameability game for proofs of membership attempts to create an honest user, we will behave normally, except for one honest user i^* where we will set $\mathsf{pk} = X_2$. When the adversary queries the oracle for proofs of memberships for this user i^*, we will set $I = a\mathsf{pk}$, which is distributed correctly, generate $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$ as normal for $\sigma^1 = (A, B, C)$ and simulate the proof, which is possible due to the zero knowledge property. The adversary in the non–frameability game for proofs of membership will output two proofs of membership that individually pass verification, but fail together: one generated honestly by the oracle and one that was not output by the oracle. Assume that the proof of membership output from the oracle, was that of the honest user i^*, which occurs with probability $1/k$, where k is the number of honest users. Then for both signatures output $I = aX_2$. We can then extract the discrete logarithm of I base H from the attached proof, due to the proof of knowledge property, which will provide the discrete logarithm of X_2 base X_1.

Chapter 8

Layered MPC [DKI+23]

8.1 Overview

The goal of this paper is to build a layered MPC protocol that takes inputs from a set of clients in the input layer and securely delivers a function of the inputs to a set of output clients in a later layer. For $t < n/3$, we present two layered protocols for general MPC with t-security: a simple but inefficient construction based on CNF secret sharing and a more complex but efficient construction based on Shamir secret sharing.

Owing to a highly restrictive communication pattern and the presence of a very powerful adversary, implementing layered MPC with optimal corruption threshold presents several interesting challenges. The most apparent is the complete prohibition of interaction, as parties executing the protocol do not persist. We emulate a limited kind of interaction by having a party who wants to speak a second time hide all possible messages it may want to convey in a future layer and selectively reveal the appropriate message to the next layer. In such cases, it is imperative to the security of the party that only the appropriate message is revealed while the other messages are effectively destroyed. Interestingly, realizing this limited form of interaction takes us a long way in implementing layered MPC. This leads us to the first primitive we construct in this presentation:

Future Messaging. Future messaging allows a party (sender) to securely send a message to another party (receiver) situated in a later layer. To send a message two layers down, the sender can secret share the message onto the next layer using any t-secure secret sharing scheme; parties in the next layer can then forward these shares to the receiver who can recover the message by robust reconstruction of the received shares. We extend this intuition to allow a sender to securely send a message to a designated receiver in any future layer. This protocol is non-commiting; hence, a corrupt sender can choose the message to deliver to the receiver based on the adversary's view until the layer in which the receiver is situated. Effectively, future messaging allows rushing till the receiver's layer! Future messaging allows a sender to distribute a secret sharing of a value onto a future layer; parties in this layer can disclose this value to a receiver (or broadcast it to all parties) in the next layer based on a unanimous decision (potentially depending on computation that was carried out in an intermediate layer). In this manner, we emulate the aforementioned (limited) interaction by the sender.

MPC using CNF Shares. Equipped with a protocol for future messaging, we set out to build a layered protocol for verifiable secret sharing (VSS). We will then follow the standard approach for secure function evaluation, where a layered arithmetic circuit computing the function is evaluated by progressively and securely computing secret shares of the value on the output wire of each gate using the secret shares of the values on the input wires, finally revealing the values on the output wires of the circuit to the output clients.

Verifiable CNF secret sharing. To achieve verifiable CNF secret sharing, it suffices to implement a seemingly simpler primitive, namely future multicast, which allows a dealer to securely send a message to a designated subset of receivers in a later output layer with the guarantee that all receivers get the same message even if the sender is corrupt. Verifiable CNF secret sharing is achieved by having the dealer split the secret into $\binom{n}{t}$ additive shares (a share r_T for each $n-t$ sized set $T \subset [n]$) and multicast r_T to all output clients in T.

While implementing multicast, we encounter many challenges inherent to layered MPC. When realizing multicast, the sender sends the same message to a (sub)set of parties in the next layer, who raise a complaint if they receive distinct messages, in which case the sender publicly discloses the message. Clearly, this sequence of interactions is non-trivial to realize in a layered network, where the sender cannot speak a second time and the parties in a layer cannot communicate with each other. Hence, we use a weak notion of secure addition (See Section 8.3.2) to allow the receiving parties to securely reveal the difference between the values they received to all parties two layers down. If the difference is non-zero for any pair of values, the layer that learns this difference collectively decides to disclose the sender's message using the trick we previously outlined.

Having implemented verifiable CNF secret sharing, we proceed to secure computation of arithmetic gates. Since the secret sharing is linear, addition and multiplication-by-constant gates can be computed by local processing, which leaves us with the secure computation of the multiplication gate that takes the secret shares of two values and computes a secret sharing of their product.

Multiplication. Our layered protocol for multiplication is built by porting the classic protocol for secure multiplication in the standard (non-layered) setting. In this process, we face all the challenges we encountered while realizing future multicast. Suppose a value is secret shared on a layer and is also required in another layer. Naively replicating the same share in the later layer is insecure since the adversary can reconstruct the secret by corrupting t parties in each of these layers and obtaining $2t$ shares. We get around this problem with a simple trick that avoids using a full-fledged protocol for resharing CNF shares.

We realize secure computation by evaluating a layered arithmetic circuit using the protocols we constructed so far. To properly process the layered circuit, we rely on the invariant that the secret shares of the values on all the input wires to any layer of the circuit are simultaneously available on the same layer of the layered network. However, secret shares of the output of a linear gate (addition or multiplication-by-constant) can be computed locally while those of a multiplication gate using our protocol consume several layers. To keep the invariant, we need the outputs of the linear gates to be available on the output

layer of multiplication. Once again, the shares of the outputs cannot be naively secret shared. Instead, we attach a multiplication gate to the output wire of linear gate that takes identity as the other input; this ensures that the shares of the values on all output wires are available simultaneously on the same layer.

Composability of layered protocols. We use simpler layered protocols as subroutines for building more complex ones. For example, the multiplication protocol uses a protocol for verifiable secret sharing (among others) as a subroutine. Hence, it is necessary that the concurrent execution of layered protocols preserve their security guarantees under concurrent composition. We refrain from first proving UC security of our building blocks and then using modular composition theorems since such an analysis will be cumbersome over a synchronous layered graph. Instead, we prove the security of our protocols by constructing simulators and carefully arguing their security. We establish game based properties of layered protocols that are preserved when they are used as subroutines and prove the security using hybrid arguments that exploit these properties. Finally, a few of our constructions make exclusively sequential (non-concurrent) calls to subroutines that have been proven to be standalone secure; in such instances, we use the sequential composition theorem of Canetti [Can00] to argue security (see the security proofs for future messaging and secure function evaluation protocols).

Efficient MPC using Shamir Secret Sharing. We build layered protocols whose communication complexity scales polynomially with the number of parties per layer. This is achieved by porting the cannonical secure function evaluation protocol using Shamir secret shares into the layered model. To achieve this, we first develop a layered protocol for verifiable Shamir secret sharing.

Verifiable Shamir secret sharing. We "port" the classic protocol for VSS in the standard setting to the layered setting using the tools we developed in the previous sections along the way to tackle the usual challenges faced in the process. At the end of this process, the parties in the layer right after the input layer hold the purported shares of the dealer's secret and parties 5 layers down publicly hold the updates to the purported shares such that, they together form a valid secret sharing. The parties cannot transfer these shares to the shareholders in the output layer without causing duplication. To get around this, the dealer secret shares coefficients of a random degree-t polynomial they wish to use for Shamir secret sharing; the evaluation of the polynomial at distinct points is computed using linear operations and securely delivered to the shareholders in the output layer. This ensures privacy of the secret when the dealer is honest.

Equipped with a layered protocol for Shamir VSS, we use known techniques to realize resharing which allows a layer holding valid shares of a value to securely deliver fresh shares of the same value to a later layer. Using VSS and resharing, porting protocols for secure multiplication and then secure function evaluation into the layered setting is relatively straightforward. We depart form the protocol for general MPC provided in [CDN15]. The protocol uses a form of reinforced secret sharing where the shares of a secret are further secret shared among the shareholders, which is straightforward to implement using VSS and resharing.

8.2 Layered MPC

A layered MPC protocol can be viewed as a special case of standard MPC with a general adversary structure, specialized in the following way: (1) the interaction pattern is defined by a layered graph; (2) the adversary can corrupt at most t parties in each layer. This is illustrated in Fig. 8.1 and formalized below.

Definition 8.1 (Layered MPC). *Let n, t, d be positive integers. An (n, t, d)-*layered protocol *is a synchronous protocol Π over secure point-to-point channels and a broadcast channel, with the following special features.*

- ***Parties.*** *There are $N = n(d+1)$ parties partitioned into $d+1$ layers $\mathcal{L}_i$, $0 \le i \le d$, where $|\mathcal{L}_i| = n$. Parties in the first layer $\mathcal{L}_0$ and the last layer $\mathcal{L}_d$ are referred to as* input clients *and* output clients*, respectively.*
- ***Interaction pattern.*** *The interaction consists of d rounds, where in round i parties in $\mathcal{L}_{i-1}$ may send messages to parties in $\mathcal{L}_i$ over secure point-to-point channels. By default, we additionally allow each party in $\mathcal{L}_{i-1}$ to send a broadcast message to all parties in $\mathcal{L}_i$.*
- ***Functionalities.*** *We consider functionalities f that take inputs from input clients and deliver outputs to output clients.*
- ***Adversaries.*** *We consider adversaries who may corrupt any number of input and output clients, and additionally corrupt t parties in each intermediate layer $\mathcal{L}_i$, $0 < i < d$. We consider active, rushing, adaptive[1] adversaries.*

We say that a protocol Π is a layered MPC protocol for f *if it realizes f in the standard sense of (standalone) secure MPC with general adversary structures [Can00, Gol09, HM00]. We require* perfect full security *(with guaranteed output delivery).*

Remark 8.1 (Generalized layered MPC). *The above definition is meant to give the simplest formalization of the core problem we study. It can be naturally extended to allow a different number of parties n_i and a different corruption threshold t_i in each layer (our main feasibility result extends to the case where $t_i < n_i/3$), and to allow inputs and outputs from parties in intermediate layers. Our strict notion of perfect full security can also be relaxed in the natural ways. In some cases, we will present efficiency improvements that achieve* computational *(full) security with perfect correctness, meaning that the effect of a computationally unbounded adversary on the outputs of honest parties can be perfectly simulated.*

The need for ideal broadcast: In Section 8.8 we show that broadcast for layered MPC is impossible if $t > 0$. Hence, we must assume ideal broadcast.

[1] In the coming sections our security analysis is with respect to non-adaptive adversaries for simplicity. In Section 8.2.2 we justify this leap appealing to the work of [CDD+04].

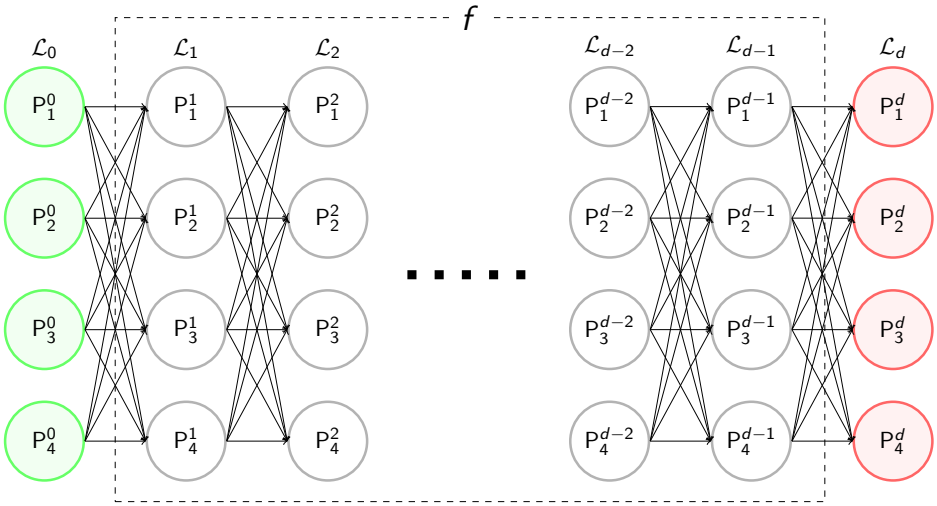


Figure 8.1: Layered MPC computing function f with $n = 4$

8.2.1 Security in Layered MPC Implies Proactive Security

The original definition of a Mobile Adversary [OY91] gives the adversary t pebbles at the outset of the protocol. It can then place the pebbles freely at the beginning of every round among the parties involved where a pebble represents fully corrupting a party. If a pebble is removed from a party, at the beginning of next time period the party will "reboot" into a pre-specified state and its random tape will be renewed. As such, this pebble game represents the race between corruption and recovery and results in a system where a bound is given on the number of corrupted parties which holds in each time period, but in each period the set of corrupted parties can change. We now define the mobile adversary and the execution of a protocol in the context of a mobile adversary.

Definition 8.2 (Mobile Adversary). *A (t, ρ)-mobile adversary, with corruption threshold t and roaming speed (mobility) ρ, is an adversary that can corrupt at most t parties in each time period where a time period consists of ρ rounds. If $\rho = 1$, we say that the adversary has maximal roaming speed or is a maximally mobile t-adversary.*

Definition 8.3 (Maximally Proactive Security). *Let r be the round number and T_r the set of at most t parties that is corrupted in round r. In each round the parties can communicate using pairwise secure point-to-point channels or broadcast. An execution of a protocol Π with R rounds of communication in the presence of a maximally mobile t-adversary proceeds as follows.*

(0) All parties receive inputs. Adversary chooses an initial set T_0 of t parties to corrupt.

(1) Initialize a round counter $r = 1$.

(2) Parties send messages of round r (including to themselves), and honest parties update their state.

(3) Adversary chooses a new set T_r of t parties to corrupt.

(4) *Parties receive messages of round r.*

(5) *If $r < R$, increment r and go to step (2). Otherwise, honest parties compute an output from messages received in round R.*

A protocol that is secure when executed in the presence of a maximally mobile t-adversary is called maximally proactive.

Remark 8.2 (Maximal Mobility with Rushing). *We note that a stronger notion of maximally mobile adversary is conceivable; one who corrupts at most t parties in each round but rushes in each round. An execution of a protocol in the presence of such an adversary proceeds as described in Definition 8.3 with one notable difference: in step (2), the adversary receives the messages in round r sent by all honest parties to every party in T_{r-1}, and then chooses the messages sent by each party in T_{r-1} in round r to all parties. In this setting, in each round r, the adversary effectively learns the messages received by parties in $T_{r-1} \cup T_r$, but actively corrupts only parties in T_{r-1}. Thus, achieving security in the presence of such and adversary is at least as hard as achieving security against a standard (non-mobile) mixed adversary that corrupts t parties actively and t additional parties passively. Fitzi, Hirt and Maurer showed in [FHM98] that, in this setting, VSS and MPC is feasible if and only if, respectively, $t < n/4$ and $t < n/5$. Whether these thresholds carry over to maximally proactive setting with rushing for VSS and MPC is left as an open problem.*

We wish to characterize the relationship between layered and maximally proactive security. Intuitively it is clear, that each layer in the context of layered MPC represents a new round in a maximally proactive protocol and the maximally mobile adversary corrupts a (possibly) new set of parties in every round. In Lemma 8.1 we capture this intuition in a more formal way. Of course, a necessary assumption is the protocol's ability to include special erasure instructions whereby parties remove sensitive data from the their local state.

Lemma 8.1 (Layered and Maximally Proactive Security). *Secure layered MPC implies secure maximally proactive MPC under the assumption of secure erasures.*

Proof. Let Π be an (n, t, d)-layered protocol for computing the functionality f with $N = n(d+1)$ parties partitioned into layers $\mathcal{L}_0, \ldots, \mathcal{L}_d$ (Definition 8.1). And let $\mathcal{A}_{\text{mobile}}$ be a maximally mobile t-adversary (Definition 8.2). We prove the implication by constructing a simulator $\mathcal{S}$ that perfectly emulates $\mathcal{A}_{\text{mobile}}$, effectively reducing security for maximally proactive MPC to security for layered MPC.

First, we argue that the interaction pattern induced by layered MPC is equivalent to that of the virtual model (transmission graph) of maximally proactive MPC. Consider the parties $\mathcal{P}_1^0, \ldots, \mathcal{P}_n^d$ computing Π and assume a maximally proactive setting with n parties labeled as $\mathsf{Q}_1, \ldots, \mathsf{Q}_n$. Due to the assumption of secure erasure, we can associate a virtual party Q_i^r with each round $0 \leq r < d$. Unless, Q_i is corrupted in both round $r-1$ and r, the party Q_i^{r-1} shares no state with Q_i^r apart from the messages received over broadcast or secure point-to-point channels. Similarly, from the definition of layered MPC $\mathcal{P}_i^r$ shares

no state with $\mathcal{P}_i^{r-1}$ unless both are corrupted. Finally, we observe that the interaction pattern between parties $\mathcal{P}_1^0, \ldots, \mathcal{P}_n^d$ is equivalent to the interaction pattern induced by the virtual model connecting the virtual parties $\mathsf{Q}_1^0, \ldots, \mathsf{Q}_n^d$.

We now sketch the simulator $\mathcal{S}$. Let $\mathcal{A}_{\text{layered}}$ be the following (n, t, d)-layered MPC adversary during an execution of protocol Π. $\mathcal{A}_{\text{layered}}$ runs $\mathcal{A}_{\text{mobile}}$ internally and sets $r = 0$ before the first round starts. Then, $\mathcal{A}_{\text{layered}}$ does the following, for each $0 \leq r \leq d$: (1) it receives from $\mathcal{A}_{\text{mobile}}$ a set of parties to corrupt $T_r = \{\mathsf{Q}_0^r, \ldots, \mathsf{Q}_t^r\}$ (2) it corrupts corresponding parties $\{\mathcal{P}_0^r, \ldots, \mathcal{P}_t^r\}$ in the execution of Π. (3) it returns the state of the corrupted parties in round r to $\mathcal{A}_{\text{mobile}}$.

Since Π is a protocol for layered MPC there exists a simulator $\mathcal{S}_{\text{layered}}$ for $\mathcal{A}_{\text{layered}}$ which we will use to construct our final simulator $\mathcal{S}$. $\mathcal{S}$ runs $\mathcal{S}_{\text{layered}}$ internally in the following way for each round $0 \leq r \leq d$. (1) $\mathcal{S}$ receives from $\mathcal{S}_{\text{layered}}$ a request to corrupt parties $\{\mathcal{P}_0^r, \ldots, \mathcal{P}_t^r\}$. (2) $\mathcal{S}$ sends to the functionality a request to corrupt $T_r = \{\mathsf{Q}_0^r, \ldots, \mathsf{Q}_t^r\}$. (3) $\mathcal{S}$ provides the state of the parties from the functionality to $\mathcal{S}_{\text{layered}}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{S}_{\text{layered}}$ outputs. □

We note that while Lemma 8.1 characterizes a strong relation between the layered MPC model and security in the presence of a maximally mobile adversary, the existing literature generally considers proactive security against a slower-moving adversary. In [ADN06, BELO15, MZW+19, ELL20], the protocol time-line is split into phases where each protocol round belongs to exactly one phase and between each pair of consecutive phases a refresh protocol is run to ensure re-randomization and redistribution of the secret. Typically, the adversary can then adaptively corrupt at most t parties between the start of one refresh until *the end* of the next, effectively, halving the mobile adversary's corruption budget during the run of a refresh protocol. Finally, the assumed mobility of the adversary often, somehow conveniently, aligns with the round complexity of computing a single layer of the layered circuit.

We do not make such assumptions about the maximally mobile adversary and, as such, designing secure protocols for maximally proactive MPC is significantly more challenging.

Definition 8.4 (Maximally Proactive MPC). *If Π is a protocol that securely (with erasures) computes any functionality f while executing in the presence of a maximally mobile adversary. Then, Π is a protocol for maximally proactive MPC.*

A protocol for maximally proactive secret sharing is an instance of maximally proactive MPC that allows a dealer to share a secret s among a group of n parties such that the secret remains secure against a maximally mobile adversary and allows the final shareholder set of n parties to open the secret. A refresh protocol prevents the adversary from discovering and destroying the secrets.

Definition 8.5 (Maximally Proactive Secret Sharing). *A maximally proactive secret sharing protocol is a set of instances of maximally proactive MPC, each associated with a* phase *and executing algorithms of a robust secret sharing scheme S. The initial phase is the*

Share *phase, then a sequence of* Refresh *phases are executed, and finally the* Open *phase. Each phase is described below.*

- Share. *The instance of maximally proactive MPC for the* Share *phase has a designated input client $\mathcal{P}_D$ (the dealer) giving secret s as input and output clients $\mathcal{P}_1, \dots, \mathcal{P}_n$ receiving shares $s_1, \dots, s_n$ as output. In this phase, the maximally proactive MPC instance executes the* Sh *algorithm of the robust secret sharing scheme on input s to obtain $s_1, \dots, s_n$.*

- Refresh. *In the refresh phase maximally proactive MPC takes as input shares $s_1, \dots, s_n$ from the parties $\mathcal{P}_1, \dots, \mathcal{P}_n$ and outputs new and independent shares $\hat{s}_1, \dots, \hat{s}_n$ to the same parties such that if $\mathsf{Rec}(\{s_i\}_{i\in[n]}) = s$, then $\mathsf{Rec}(\{\hat{s}_i\}_{i\in[n]}) = s$. That is, the MPC executes $\mathsf{Sh}(\mathsf{Rec}(\{s_i\}_{i\in[n]}))$ of S under fresh randomness and securely distributes the resulting shares to $\mathcal{P}_1, \dots, \mathcal{P}_n$.*

- Open. *The final phase involves all n parties broadcasting their shares. Then, all honest parties run the reconstruction algorithm* Rec *of the underlying scheme S on the set of shares.*

8.2.2 Adaptivity and Composability in Layered MPC

Let Π_g be a layered protocol realizing functionality g with standalone t-security, and let Π_f be another layered protocol in which Π_g is used as a subroutine to implement g. Suppose the layers where g is computed using Π_g do not execute any other protocol in parallel; i.e., only a single invocation of Π_g is made in such layers. Then, to prove the security of Π_f, it is sufficient to show that Π_f is t-secure in the so called g-hybrid model, where the calls to the sub routine Π_g is replaced with calls to the functionality g itself. This allows for a modular construction and analysis of protocols.

Formally, the g-hybrid model involves a communication protocol as well as calls to functionality g. Suppose I is the designated output layer of g. In a protocol Π_f in g-hybrid model, parties in layer $i-1$ can send their inputs to functionality g in round i. The functionality will deliver the output of g to receivers in the output layer I in round I which may be used by the parties in executing Π_f.

The following proposition adapts the sequential composability theorem of [Can00] to the layered setting. The proposition holds simply because a layered protocol with d layers and n parties per layer is essentially a nd party protocol with communication between a pair of adjacent layers in every round.

Proposition 8.1 (Sequential Composability for layered protocols). *Suppose a (n, t, d)-layered protocol Θ implements a functionality g with perfect standalone t-security [Can00, Gol09]. Suppose a layered protocol Π with input layer $\mathcal{L}_0$ and output layer $\mathcal{L}'_{d'}$, $d' > d$ invokes Θ as a subroutine from $\mathcal{L}_a$ to $\mathcal{L}_{a+d}$, where $0 \le a < a + d \le d'$. Π making subroutine calls to Θ is t-secure if it is t-secure in the g-hybrid model.*

Universal Composability. As discussed in Definition 8.1, we are interested in realizing functionalities f that take input from the input clients in layer $\mathcal{L}_0$ by default and deliver outputs to the output clients in the last layer (layer $\mathcal{L}_d$) of a layered network. We develop a protocol for computing general functionalities in the stand-alone model showing perfect security by means of a straight-line black-box simulator and, thus, we can invoke Theorem 1.2 in [KLR10] and argue that the protocol is, in fact, secure under the definition of universal composability[2].

On Adaptive Adversaries. In Definition 8.1, we define layered MPC in the presence of a *rushing* and *adaptive* adversary. Clearly, this extra power for the adversary separates layered MPC from maximally proactive MPC and shows that layered MPC is strictly stronger. Looking forward, we will, however, only analyze the layered protocols with respect to static (and rushing) adversaries. To argue adaptive security, we need to be able to simulate even when the real world adversary corrupts a party midway through the protocol. [CDD+04] showed an exotic example of a perfectly secure protocol with static security against malicious adversaries but without adaptive security. Fortunately, all our protocols are based on linear secret sharing which makes extending our analysis to layered (*and* adaptive) MPC significantly easier.

As an example, consider a simulator's job when a set of parties $\mathcal{C}$ is already corrupted during a protocol execution and a new party $\mathcal{P}_i$ has just been added to this set. First, the simulator needs to construct a complete view (including the input) of the honest $\mathcal{P}_i$ that is consistent with all messages exchanged with the ideal functionality and communication with parties in $\mathcal{C}$. Secondly, the simulator's state needs to be "extended" with this new information. Concretely, the state should be as if $\mathcal{P}_i$ has been corrupted from the start of the protocol but behaved honestly until this point. In our protocols for perfect layered MPC, we let the simulator handle this challenge using conditional sampling. Since parties in $\mathcal{C}$ will only hold shares of a linear secret sharing scheme, even if the newly corrupted $\mathcal{P}_i$ is the dealer of such shares we can simulate the randomness used in the sharing algorithm. This is feasible since as long as the shares of $n - t$ honest parties are fixing the secret, the simulator is free to change the randomness to be consistent with the shares of parties in $\mathcal{C}$. Finally we note that when referring to computationally secure (PRG-based) protocols, we either need to settle for non-adaptive security or implement the PRG in the random oracle model.

8.3 Basic Primitives

We introduce the basic primitives Future Messaging (f_{FM}) and Multiparty Addition (f_{Add}) that that serve as building blocks for later constructions. In the layered model, Future Messaging is a primitive which allows an input client S to securely send a message m

[2] While we can meaningfully argue that the final protocol for computing general functionalities is UC-secure, we do not treat individual components of this protocol in a UC manner. This would require a significant modelling effort of communication and synchronization for layered MPC and would be counterproductive in our effort to present layered MPC as a simple special case of secure MPC as in [Can00, Gol09].

to an output client R in a later layer. Multiparty Addition allows a subset of parties in a layer to broadcast the sum of their inputs to all parties in a later layer.

8.3.1 Future Messaging

Future Messaging emulates a *secure channel* between a sender S and a receiver R in a future layer. As such, the primitive is similar[3] to Secure Message Transmission (SMT) over the specific directed and layered network where intermediate nodes may take part in the protocol and not merely forward messages from adjacent nodes. The functionality is formalized in Fig. 8.2.

Future Messaging functionality f_{FM}

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, receiver $\mathsf{R} \in \mathcal{L}_d$ for $d > 0$ and message domain M.
SECRET INPUTS: S has input $m \in M$.

f_{FM} receives m from S, and delivers m to R.

Figure 8.2: Future Messaging functionality f_{FM}

8.3.1.1 Parallel Composition.

Functionality f_{FM} delivers a message from a sender to a receiver in a later layer $\mathcal{L}_d$. However, when our protocol implementing f_{FM} is composed in parallel, the resulting functionality is not the natural parallel composition of f_{FM} which takes the input from each sender to each receiver and delivers them.

In fact, this functionality is impossible to realize even in the trivial case of messaging from one layer to the very next using the provided secure communication link. As an example, suppose communication from $\mathsf{S}_1 \in \mathcal{L}_0$ to $\mathsf{R}_1 \in \mathcal{L}_1$ and from $\mathsf{S}_2 \in \mathcal{L}_0$ to $\mathsf{R}_2 \in \mathcal{L}_1$ are composed in parallel. Now, a rushing adversary corrupting S_1 and R_2 can collect the message from S_2 to R_2 and set this as the message from S_1 to R_1. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. See Remark 8.4 in Section 8.9 for more details.

We capture the functionality realized by parallel execution of our future messaging protocol using a corruption aware functionality in Figure 8.3.

[3] The instance of Future Messaging with honest sender in $\mathcal{L}_0$ and honest receiver in $\mathcal{L}_2$ is equivalent to perfect 1-way SMT.

Corruption-aware parallel Future Messaging functionality f_{FM}^n

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 0$. The domain $M_{i,j}$ of message from S_i to R_j.
SECRET INPUTS: Each S_i wants to send each R_j a message $m_{(i,j)} \in M_{i,j}$.
ADDITIONAL INPUT TO FUNCTIONALITY: Set of corrupted parties $\mathcal{I}_0 \subseteq \mathcal{L}_0$ and corrupted receivers $\mathcal{I}_d \subseteq \mathcal{L}_d$.

1. For each honest $\mathsf{S}_i \notin \mathcal{I}_0$ and each $\mathsf{R}_j \in \mathcal{L}_d$, f_{FM}^n receives message $m_{(i,j)}$ from S_i to R_j.
2. For each honest $\mathsf{S}_i \notin \mathcal{I}_0$ and corrupt $\mathsf{R}_j \in \mathcal{I}_d$, f_{FM}^n forwards $m_{(i,j)}$ to the (ideal) adversary.
3. For each corrupt $\mathsf{S}_i \in \mathcal{I}_0$ and each $\mathsf{R}_j \in \mathcal{L}_d$, f_{FM}^n receives from the (ideal) adversary the message $m_{(i,j)}$ that S_i wants to send to R_j.
4. For each $\mathsf{S}_i \in \mathcal{L}_0$ and $\mathsf{R}_j \in \mathcal{L}_d$, f_{FM}^n sends $m_{(i,j)}$ to R_j as message from S_i.

Figure 8.3: Corruption-aware parallel Future Messaging functionality f_{FM}^n

8.3.1.2 A Protocol for Future Messaging.

Realizing Future Messaging from a sender in $\mathcal{L}_0$ to a receiver in $\mathcal{L}_1$ is trivial since there is a secure communication link between any such pair.

A $(n, t, 2)$-layered protocol for Future Messaging from a sender in $\mathcal{L}_0$ to a receiver in $\mathcal{L}_2$ can be achieved as follows. Sender $\mathsf{S} \in \mathcal{L}_0$ shares the message m among the parties in $\mathcal{L}_1$ using a t-secure robust secret sharing scheme. The parties in $\mathcal{L}_1$ forward their shares to the receiver $\mathsf{R} \in \mathcal{L}_2$ who uses the reconstruction algorithm on the received shares to recover the message. By t-security of the secret sharing scheme, an adversary corrupting at most t parties in $\mathcal{L}_1$ learns nothing about the message. However, since the secret sharing scheme is t-robust, R correctly reconstructs m even if at most t corrupt parties send incorrect shares.

This idea can be generalized to construct Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_d$ for any $d > 2$ using the secure (n, t, ℓ)-layered protocol for Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and then from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. Here, ℓ is any number such that $0 < \ell < d$; specifically, we can take $\ell = \lfloor \frac{d}{2} \rfloor$. This is achieved as follows. The sender $\mathsf{S} \in \mathcal{L}_0$ produces shares $(s_1, \ldots, s_n)$ of its message m, and sends the share s_i to the i-th party $(\mathcal{P}_i^\ell)$ in $\mathcal{L}_\ell$ using Future Messaging from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. Each party in level ℓ forwards its share to the receiver using Future Messaging from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

This protocol can be executed in parallel, for each sender in $\mathcal{L}_0$ and receiver in $\mathcal{L}_d$, in order to realize the corruption aware (parallel) functionality f_{FM}^n (Figure 8.3) from $\mathcal{L}_0$ to $\mathcal{L}_d$ using f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. The protocol is formally described in Figure 8.4.

Π^n_{FM}, an (n, t, d)-layered protocol realizing f^n_{FM}

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \ldots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 1$.
SECRET INPUTS: Each S_i wants to send $m_{(i,j)} \in M$ to a each receiver R_j.
RESOURCES: f^n_{FM} (with message domain M^n) from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

1. Each $\mathsf{S}_i, i \in [n]$ samples $(s_{(i,j),1}, \ldots, s_{(i,j),n}) \leftarrow \mathsf{Sh}(m_{(i,j)})$ for each $j \in [n]$.
2. For $k \in [n]$, S_i sets the message to $\mathcal{P}^\ell_k \in \mathcal{L}_\ell$ in f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ to $(s_{(i,1),k}, \ldots, s_{(i,n),k})$.
3. Each party $\mathcal{P}^\ell_k : k \in [n]$ receives $(\hat{s}_{(i,1),k}, \ldots, \hat{s}_{(i,n),k})$ from $\mathsf{S}_i, i \in [n]$ (delivered by f^n_{FM}).
4. $\mathcal{P}^\ell_k, k \in [n]$ sets the message to $\mathsf{R}_j \in \mathcal{L}_d$ in f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ to $(\hat{s}_{(1,j),k}, \ldots, \hat{s}_{(n,j),k})$.
5. Each receiver $\mathsf{R}_j : j \in [n]$ computes the message from $\mathsf{S}_i : i \in [n]$ as $\mathsf{Rec}(\hat{s}_{(i,j),1}, \ldots, \hat{s}_{(i,j),n})$.

Figure 8.4: Layered protocol Π^n_{FM}

Lemma 8.2 (Layered protocol for f^n_{FM})**.** *Let* $(\mathsf{Sh}, \mathsf{Rec})$ *be a robust* (t, n) *secret-sharing scheme Definition 5.1, the* (n, t, d)*-layered protocol in Figure 8.4 realizes the functionality* f^n_{FM} *in Figure 8.3 with perfect security for* $t < n/3$.

We formally describe the simulator and provide a formal proof in Section 8.9.2.

Going forward, we will focus on the (non-parallel) Future Messaging functionality f_{FM} (Fig. 8.2) from a designated sender in a layer to a designated receiver in a later layer. This is, indeed, a special case of f^n_{FM} $(n = 1)$ and a protocol was outlined informally in the beginning of this section.

Theorem 8.1. *For any* $d > 0$*, and message domain* M*, there exists an* (n, t, d)*-layered protocol* Π_{FM} *that realizes* f_{FM} *from a sender in* $\mathcal{L}_0$ *to a receiver in* $\mathcal{L}_d$ *with communication complexity* $O(n^{\lceil \log d \rceil} \log |M|)$.

Proof. For $d = 1$, there is a trivial protocol that realizes f_{FM} in which the sender sends the message (from a domain M) directly to the receiver using the provided secure communication link. The communication complexity of realizing this is simply $\log |M|$.

Suppose $d > 1$ and $\ell = \lfloor \frac{d}{2} \rfloor$. Consider protocols Π and Π' that realize functionalities f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and from $\mathcal{L}_\ell$ to $\mathcal{L}_d$, respectively for message domain M^n. In the protocol in Figure 8.4, the f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ and f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ are called, sequentially. Hence, by the sequential modular composition theorem for layered protocols in Proposition 8.1, the protocol obtained by replacing these oracle calls with subroutine calls to Π and Π', is secure against any layered adversary that corrupts at most t parties in layers 1 to $\ell - 1$ and $\ell + 1$ to $d - 1$ in addition to corrupting at most t parties in layer ℓ. The communication complexity of the resulting protocol is the sum of communication complexity of Π and Π'. The statement of the theorem is obtained by recursion using this observation and the existence of the trivial protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_1$. □

Corollary 8.1. *Suppose* Π_{FM} *is a* (n, t, d)*-layered protocol realizing* f_{FM} *from a sender* $\mathsf{S} \in \mathcal{L}_0$ *to a receiver* $\mathsf{R} \in \mathcal{L}_d$*. The following statements hold when* Π_{FM} *is executed in the presence of any adversary* $\mathcal{A}$ *described in Definition 8.1:*

(a) If S *is honest,* R *correctly recovers the input of* S *at the end of* Π_{FM}*.*

(b) When S *and* R *are honest, and for any pair of inputs* $m, m' \in M$*,*

$$\mathsf{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}}(m) \equiv \mathsf{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}}(m').$$

8.3.2 Multiparty Addition

The Multiparty Addition functionality f_{Add} takes inputs from a set of input clients and delivers the sum of the inputs to all output clients in $\mathcal{L}_2$. However, f_{Add} allows the adversary to choose the inputs of corrupt input clients after learning the sum of the inputs of the honest clients. Hence, if at least one party with input to f_{Add} is corrupt, the adversary can choose the value that f_{Add} outputs. Note that, this necessarily makes f_{Add} a corruption aware functionality. The functionality is formally defined in Fig. 8.5 and can be realized by an $(n, t, 2)$-layered protocol as outlined below.

Multiparty Addition functionality f_{Add}

PUBLIC PARAMETERS: Input clients $S \subseteq \mathcal{L}_0$, output clients $\mathcal{L}_2$, input domain is some finite group $\mathbb{G}$.
SECRET INPUTS: Each $\mathsf{S}_i \in S$ has input $x_i \in \mathbb{G}$.
ADDITIONAL INPUT: Set of corrupt input clients $\mathcal{I} \subseteq S$.

1. f_{Add} receives input x_i from each honest $\mathsf{S}_i \in S$.
2. If $\mathcal{I} \neq \emptyset$, f_{Add} leaks the sum of the inputs of honest S_i to the (ideal) adversary and receives a value $y \in \mathbb{G}$ from the adversary.
3. To all output clients in $\mathcal{L}_2$, f_{Add} delivers y if $\mathcal{I} \neq \emptyset$ and $\sum_{i:\mathsf{S}_i \in S} x_i$ otherwise.

Figure 8.5: Multiparty Addition functionality f_{Add}

Each party in $\mathsf{S}_i \in S$ secret shares its input x_i to the parties in next layer using a t-robust linear secret sharing scheme. Parties in $\mathcal{L}_1$ broadcasts the sum of their respective shares for each of the inputs. Each party in $\mathcal{L}_2$ recovers the output by running the reconstruction algorithm on the received sum of shares. A formal description of the protocol is presented in Fig. 8.6.

Π_{Add}, an $(n, t, 2)$-layered protocol for f_{Add}

PUBLIC PARAMETERS: Input clients $\mathcal{P}_i^0$, $i \in S$, output clients $\mathcal{L}_2$, input domain is some finite group $\mathbb{G}$.
Robust (n, t) secret-sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ with secret domain M.
SECRET INPUTS: Each $\mathsf{S}_i \in S$ has input $x_i \in \mathbb{G}$.

1. Each $\mathcal{P}_i^0$, $i \in S$ samples $(x_{i,1}, \ldots, x_{i,n}) \leftarrow \mathsf{Sh}(x_i)$, and sends $x_{i,j}$ to $\mathcal{P}_j^1$ for each $j \in [n]$.
2. Each $\mathcal{P}_j^1$, $j \in [n]$ broadcasts $y_j = \sum_{i \in S} x_{i,j}$ to $\mathcal{L}_2$.
3. Each party outputs $\mathsf{Rec}(y_1, \ldots, y_j)$. We use a reconstruction function that outputs a valid element of $\mathbb{G}$ even when $y_1, \ldots, y_n$ is not a valid secret sharing with at most t corruptions.

Figure 8.6: Layered protocol Π_{Add}

Clearly, all honest parties output the same value at the end of the protocol, irrespective of the number of corruption in S. If all parties in S are honest, each party in $\mathcal{L}_2$ receives a share of $\sum_{\mathsf{S}_i \in S} x_i$ for each party in $\mathcal{L}_1$. Although corrupt parties in $\mathcal{L}_1$ can potentially send invalid shares, by t-robustness of the secret sharing scheme all honest parties in $\mathcal{L}_2$ correctly reconstruct the sum of the inputs. Finally, the adversary who corrupts a non-empty set of parties in $\mathcal{L}_2$ only learns the sum of the shares of the honest parties' inputs. Since the secret sharing scheme is linear, this would only reveal the sum of the honest parties' inputs.

The following lemma formally states the game based security guarantees of any (n, t, d)-layered protocol realizing Multiparty Addition as per above.

Lemma 8.3. *The following statements hold when an (n, t, d)-layered protocol realizing f_{Add} is executed in the presence of any adversary $\mathcal{A}$ described in Definition 8.1:*

1. *All honest clients output the same value at the end of Π_{Add}. If all input clients are honest, this value coincides with the sum of the inputs.*

2. *The view of $\mathcal{A}$ only reveals the sum of the inputs of the honest parties.*

8.4 Layered MPC based on CNF Secret Sharing

In this section, we start by building a protocol for Future Multicast based on primitives from Section 8.3. The protocol is then used in a simple way to obtain VSS using CNF-shares. We will build on this VSS protocol in order to realize secure multiplication and, finally, a protocol for layered MPC for *any* function.

8.4.1 Future Multicast

Future Multicast f_{FMcast} allows a sender S to send a secret to a set of receivers R located in a later layer. It guarantees that all honest receivers output the same value even if the sender is corrupt; if the sender is honest, this value coincides with the sender's input. Finally, if all receivers (and the sender) are honest, the secret remains hidden from the adversary. This primitive will be the backbone of our layered VSS protocol. Standard (Secure) Multicast is often described as the simplest non-trivial example of secure computation. Also, in layered MPC, Future Multicast generalizes Future Messaging and Future Broadcast[4] but is substantially harder to realize. The functionality is described in Fig. 8.7.

Future Multicast functionality f_{FMcast}

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, receiving set of parties $R \subseteq \mathcal{L}_d$, $d \geq 5$, message domain M.
SECRET INPUTS: S has input $m \in M$.

f_{FM} receives m from S, and delivers m to all parties in R.

Figure 8.7: Future Multicast functionality f_{FMcast}

8.4.1.1 A protocol for Future Multicast.

As a first step towards realizing f_{FMcast}, we construct a protocol that achieves a weaker notion of Future Multicast. In this protocol, sender S in layer $\mathcal{L}_0$ sends a share to a set of intermediaries $U_T = \{\mathcal{P}_i^1 : i \in T\} \subset \mathcal{L}_1$, in the next layer, who communicate it to the receivers $R \subseteq \mathcal{L}_5$. The protocol for *weak Future Multicast* provides the following guarantees which are formally stated in Lemma 8.4.

1. (*Agreement*). If a majority of the intermediaries are honest, all honest receivers output the same value at the end of the protocol even if S is corrupt; if the sender is honest, this value coincides with the sender's input.

2. (*Security*). If the sender, all the intermediaries in U_T and all the receivers are honest, a layered adversary does not learn the sender's secret.

Observe that, when $t < n/3$, each subset U_T of $n-t$ parties in $\mathcal{L}_1$ contains a strict minority of corrupt parties. Furthermore, there is at least one such set that contains only honest parties. Given these observations, realizing f_{FMcast} from the weaker notion is straight forward: For each set $U_T \subset \mathcal{L}_1$ of $n-t$ parties, S sends r_T to the receivers using parties in U_T as intermediaries, where r_T for all possible T, form an additive secret sharing of the sender's secret. When the sender is honest and each set of intermediaries has an honest majority, by (1), all r_T reach the receivers correctly. Furthermore, for one set of intermediaries U_{T^*}, by (2), r_{T^*} remains hidden from the adversary. Thus, receivers can

[4] Here, we refer to the primitive in the setting of layered MPC that ensures termination, validity and agreement among all parties located in some layer $d > 1$. Not Future Broadcast as defined in [GHK+21].

compute the sum of r_T for distinct sets T to obtain the secret, which will remain hidden from the layered adversary if all receivers are honest. Finally, by (1), the outputs of all honest receivers are consistent even if the sender is corrupt.

Weak Future Multicast. With the aid of a set of intermediaries $U_T = \{\mathcal{P}_i^1 : i \in T\} \subset \mathcal{L}_1$, weak Future Multicast can be achieved as follows: S sends the message r_T to every party in U_T. In addition, S distributes a robust secret sharing of r_T among the parties in $\mathcal{L}_3$ using Future Messaging. Every pair of intermediaries broadcasts the difference between the values they received to all parties in $\mathcal{L}_3$ using a protocol for the f_{Add} functionality. Additionally, each intermediary distributes a secret sharing of the value they received among the parties in $\mathcal{L}_4$. If the difference comes out non-zero for any pair, the parties in $\mathcal{L}_3$ effectively reveals r_T to all parties in $\mathcal{L}_4$ by broadcasting the shares of r_T that S distributed. Parties in $\mathcal{L}_4$ then forwards (using layer-to-layer broadcast) r_T to all the receivers in R. By robustness of the secret sharing scheme, parties in $\mathcal{L}_4$ recover r_T if it was secret shared properly by the sender; moreover, even if S sent invalid shares, all honest parties recover the same value. Hence, receivers recover r_T from this because at most $t < n/3$ parties in $\mathcal{L}_4$ are corrupt. If the difference is zero for every pair of intermediaries, each party in $\mathcal{L}_4$ reveals the share sent to them by every intermediary to all the receiver in R. Using these shares, each receiver reconstructs the value that was shared by each intermediary. If the difference was zero for every pair of intermediaries, then all honest intermediaries must have received the same value from S (which is r_T if S is honest). Hence, a majority of the values recovered by every receiver coincides with this value. This ensures (1). If S and all intermediaries are honest, r_T is not revealed to parties in $\mathcal{L}_4$, and, hence, is disclosed only to the receivers ensuring (2).

An $(n, t, 5)$-layered protocol for Future Multicast Π_{FMcast} is formally described in Fig. 8.8. Importantly, it includes the sub-protocol for weak Future Multicast $\Pi_{\mathsf{weak\text{-}FMcast}}$. We identify two important properties of Π_{FMcast} that will be used going forward. The properties are stated in Lemma 8.4 and a formal proof is provided in Section 8.10.1.

Lemma 8.4. *For any $T \in \mathcal{T}$, the following properties hold for any weak future multicast protocol with U_T as intermediaries when executed in the presence of any adversary $\mathcal{A}$:*

(a) There exists $\hat{r}$ such that all honest receivers in R output $\hat{r}$ at the end of the protocol. Furthermore, if S is honest, $\hat{r} = r$.

(b) If S, and all intermediaries and receivers are honest, for any $r, r' \in M$,

$$\mathsf{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathsf{ADVR}_{\Pi,\mathcal{A}}(r').$$

Theorem 8.2. *There is a secure $(n, t, 5)$-layered protocol realizing future multicast with input client S and output clients in R.*

Proof. Let $\Pi_{\mathsf{weak\text{-}FMcast}}$ be a protocol realizing weak future multicast. By statement (a) in Lemma 8.4, for every set of intermediaries $\{\mathcal{P}_i^1 : i \in T\}$, there exists $\hat{r}_T$ such that all honest receivers in R output $\hat{r}_T$ at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$. Furthermore, if S is honest,

$\hat{r}_T = r_T$, for each $T \in \mathcal{T}$. Hence, at the end of the future multicast protocol, say Π_{FMcast}, the outputs of all receivers are the same and coincides with the input of an honest S.

It remains to show that if the sender and all receivers are honest, $\mathcal{A}$ does not learn the sender's input. We sketch the intuition: Consider $T^* \in \mathcal{T}$ such that the parties U_{T^*} are all honest; such a set exists because there are at most t corruptions in each layer. By statement (b) in Lemma 8.4, view of $\mathcal{A}$ interacting with $\Pi_{\mathsf{weak\text{-}FMcast}}$ with intermediaries in U_{T^*} is independent of the input r_{T^*} of S. But then, the view of $\mathcal{A}$ in the entire protocol Π_{FMcast} does not depend on m since $(r_T, T \in \mathcal{T})$ is an additive secret sharing of m. We formally prove security of Π_{FMcast} by demonstrating a simulator $\mathcal{S}$ in Section 8.10.2. □

Π_{FMcast}, an $(n, t, 5)$-layered protocol for f_{FMcast}

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, receivers $R \subseteq \mathcal{L}_d$, where $d = 5$,
A (t, n) robust linear secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$.
DEFINITIONS: $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$ (Definition 5.2).
SECRET INPUTS: S has input $m \in M$.
SUBROUTINES: Protocol Π_{FM} realizing f_{FM} and Π_{Add} realizing f_{Add}.

1. S samples $\{r_T\}_{T \in \mathcal{T}}$ uniformly at random conditioned on $m = \sum_{T \in \mathcal{T}} r_T$. For each $T \in \mathcal{T}$, execute protocol $\Pi_{\text{weak-FMcast}}$ (described below) with $U_T = \{\mathcal{P}_i^1 : i \in T\}$ as intermediaries and r_T as input from S.
2. For each $T \in \mathcal{T}$, suppose $\hat{r}_T$ is the output of receiver $\mathcal{P}_i^5 \in R$ at the end of $\Pi_{\text{weak-FMcast}}$ with U_T as intermediaries. $\mathcal{P}_i^5$ outputs $\hat{m} = \sum_{T \in \mathcal{T}} \hat{r}_T$.

SUB-PROTOCOL: $\Pi_{\text{weak-FMcast}}$ with public input U_T and r as input from S.

(i). Layer 0:

1. S sends r to parties in $U_T \subset \mathcal{L}_1$ over the secure channel.
2. S samples $(r_1, \ldots, r_n) \leftarrow \mathsf{Sh}(r)$. For $k \in [n]$, S sends r_k to $\mathcal{P}_k^3$ using Π_{FM}.

(ii). Layer 1:

1. Denote the value received by $\mathcal{P}_j^1 \in U_T$ by r^j. For each $j, j' \in T$ such that $j < j'$, execute Π_{Add} to compute $r^j - r^{j'}$ and broadcast the result to $\mathcal{L}_3$.
2. Each intermediary $\mathcal{P}_j^1 \in U_T$ samples $(r_1^j, \ldots, r_n^j) \leftarrow \mathsf{Sh}(r^j)$ and sends r_i^j to $\mathcal{P}_i^4, i \in [n]$ using Π_{FM}.

(iii). Layer 3:

1. Each party $\mathcal{P}_k^3, k \in [n]$ recovers $\hat{r}_k$ as the output of Π_{FM} (see step (i).2). $\mathcal{P}_k^3$ broadcasts a complaint and $\hat{r}_k$ to all parties in $\mathcal{L}_4$ if $r^j - r^{j'} \neq 0$ (output of Π_{Add}) for some j, j'.

(iv). Layer 4:

1. If at least $n - t$ parties in $\mathcal{L}_3$ broadcasted a complaint, each $\mathcal{P}_i^4, i \in [n]$ computes $\hat{r} = \mathsf{Rec}(\hat{r}_1, \ldots, \hat{r}_n)$. $\mathcal{P}_i^4$ forwards the complaint to all receivers in R and sends $\hat{r}$ to all receivers.
2. Else, $\mathcal{P}_i^4$ recovers $\hat{r}_i^j$ as the output of Π_{FM} (see step (ii).2) with $\mathcal{P}_j^1$ as sender (with message r_i^j) and sends it to all receivers.

(v). Layer 5:

1. If at least $n - t$ parties in $\mathcal{L}_4$ reported a complaint, each $\mathcal{P}_i^5 \in R$ outputs the unique value that is sent by at least $n - t$ parties in $\mathcal{L}_4$.
2. Else, $\mathcal{P}_i^5$ recovers $\hat{r}^j = \mathsf{Rec}(\hat{r}_1^j, \ldots, \hat{r}_n^j)$, where $\hat{r}_i^j$ was sent by $\mathcal{P}_i^4$ and outputs the unique value $\hat{r}$ such that $\hat{r} = \hat{r}^j$ for at least $n - 2t$ distinct values of $j \in T$.

Figure 8.8: Layered protocol Π_{FMcast}

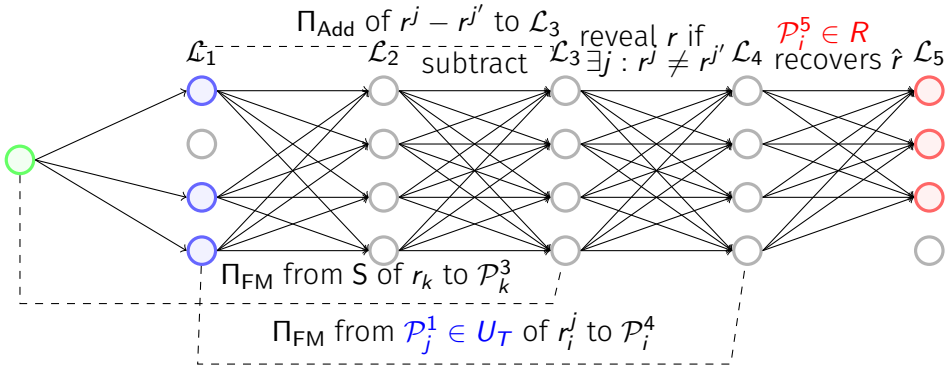


Figure 8.9: Execution flow of protocol $\Pi_{\text{weak-FMcast}}$

8.4.2 Verifiable Secret Sharing

Using future multicast presented in Section 8.4.1, realizing verifiable secret sharing (VSS) is relatively straight-forward. The sender distributes the additive shares of the secret to each set of receivers using Future Multicast. The protocol in Figure 8.10 realizes VSS from a dealer in $\mathcal{L}_0$ to shareholders in $\mathcal{L}_5$.

Π_{VSS}, an $(n, t, 5)$-layered protocol for f_{VSS}

PUBLIC PARAMETERS: Sender $\mathsf{S} \in \mathcal{L}_0$, shareholders $\mathcal{L}_5$.
DEFINITIONS: Let $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$.
SECRET INPUTS: S has input $m \in M$.
SUBROUTINES: Protocol Π_{FMcast} realizing f_{FMcast} functionality.

Layer $\mathcal{L}_0$:

1. S samples $(r_T)_{T \in \mathcal{T}}$ as additive secret sharing of m.
2. For each $T \in \mathcal{T}$, execute Π_{FMcast} with S as sender with input r_T and $\{\mathcal{P}_i^5 : i \in T\}$ as receivers.

Layer $\mathcal{L}_5$:

1. Each party $\mathcal{P}_i^5$, $i \in [n]$ recovers r_T as the output of Π_{FMcast} with S as sender if $i \in T$. $\mathcal{P}_i^5$ outputs $(r_T)_{i \in T}$ as its share.

Figure 8.10: Layered protocol Π_{VSS}

The protocol described in Figure 8.10 is a $(n, t, 5)$-layered protocol realizing VSS. This follows from the definition of Future Multicast. The following theorem proves a stronger result: Suppose n protocols are executed in parallel with $\mathcal{P}_i^0$ as dealer and $\mathcal{L}_5$ as shareholders for each $i \in [n]$, then we achieve a parallel $(n, t, 5)$-layered protocol for VSS functionality for $t < n/3$. The parallel VSS fuctionality is formally described below.

Parallel VSS functionality $f_{\text{parallel-VSS}}$

PUBLIC PARAMETERS: Senders $S_1, \ldots, S_n \in \mathcal{L}_0$, shareholders $R_1, \ldots, R_n \in \mathcal{L}_5$.
The domain M of secrets.
DEFINITIONS: Let $\mathcal{T} = \{T \subset [n] : |T| = n - t\}$.

1. Each S_i, $i \in [n]$ sends $(r_T^i)_{T \in \mathcal{T}}$ to the functionality.
2. For each $i \in [n]$ and $T \in \mathcal{T}$, functionality sends (i, T, r_T^i) to $\{\mathcal{P}_j^5 : j \in T\}$.

Figure 8.11: Parallel VSS functionality $f_{\text{parallel-VSS}}$

Theorem 8.3. *The protocol in Figure 8.10 executed in parallel realizes $f_{\text{parallel-VSS}}$ with perfect t-security for $t < n/3$ by consuming 5 layers, and by communicating $\binom{n}{t}^3 \cdot O(n^2)$ field elements over the point-to-point channels and over the broadcast channels for each secret.*

Proof. The VSS protocol is essentially several multicast protocols executed in parallel. The security of the construction follows from the security of the multicast protocol, once we ensure that the adversary cannot correlate the shares of the corrupt parties with those of the honest parties across parallel executions of multicast protocols. The simulator for multicast extracts the input of a corrupt sender in $\mathcal{L}_0$ from the view of the honest parties in the protocol up to $\mathcal{L}_4$. This allows the simulator we build for parallel VSS to extract the shares of the corrupt dealers after simulating the protocol till $\mathcal{L}_4$ and provide them to $f_{\text{parallel-VSS}}$. Whereas, a multicast from an honest sender to a set of receivers, potentially containing corrupt receivers, does not reveal the sent message to the corrupt parties until $\mathcal{L}_4$. Hence, the adversary chooses shares for corrupt parties before getting to see the shares chosen by the honest parties. This guarantees that the adversary cannot correlate the shares of the corrupted parties with the shares of the honest parties. We show a simulator and full proof in Section 8.10.3. □

8.4.2.1 Addition and multiplication-by-constant for CNF shares.

The CNF secret sharing scheme is linear; hence, parties holding valid CNF shares of a value s can locally transform it into a valid secret sharing of αs when α is a publicly known constant. In detail, let s_i be the share of s held by party i. Then, there exist $(\delta_T)_{T \in \mathcal{T}}$ such that $\sum_{T \in \mathcal{T}} \delta_T = s$, and $s_i = (\delta_T)_{T : i \in T}$ for each $i \in [n]$. Then $s_1', \ldots, s_n'$ such that $s_i' = (\alpha \delta_T)_{T : i \in T}$ is a CNF secret sharing of αs. Additionally, suppose a value r is secret shared as $(r_1, \ldots, r_n)$ where $r_i = (\gamma_T)_{T : i \in T}$ for each $i \in [n]$, and $\sum_{T \in \mathcal{T}} \gamma_T = r$. Then, $s_1'', \ldots, s_n''$ such that $s_i'' = (\delta_T + \gamma_T)_{T : i \in T}$ is a CNF secret sharing of $r + s$. In conclusion, addition and multiplication by constant of CNF shares can be computed locally.

8.4.3 Multiplication

The multiplication functionality f_{Mult} (presented in Fig. 8.12) takes valid CNF secret shares of two values r and s and computes fresh CNF secret shares of rs. This functionality requires that the input clients hold valid CNF secret sharing of the individual values to be multiplied, and that at most t input clients are corrupt. In contrast, by default, a layered adversary is allowed to corrupt arbitrarily many input and output clients.

Implementing f_{Mult}. Suppose $r_1, \ldots, r_n$ and $s_1, \ldots, s_n$ are CNF secret shares of two values r and s, respectively. Recall that, when $\mathcal{T} = \{T_1, \ldots, T_N\} = \{T \subset [n] : |T| = n - t\}$, for each $i \in [n]$, $r_i = (\gamma_j)_{j:i \in T_j}$ and $s_i = (\lambda_j)_{j:i \in T_j}$, where $\sum_{i=1}^{N} \gamma_j = r$ and $\sum_{i=1}^{N} \lambda_j = s$. To compute a secret sharing of rs, it suffices to compute the secret sharing of $\gamma_i \lambda_j$ for every $i, j \in [N]$; secret shares of rs can be computed as the sum of these secret shares, which can be obtained by local computations. This follows from the fact that, $rs = \sum_{i=1}^{N} \sum_{j=1}^{N} \gamma_i \lambda_j$.

The main challenge in implementing multiplication is in obtaining correct secret shares of $\gamma_i \lambda_j$, for all $i, j \in [N]$. In the non-layered setting, classic protocols tackle this by having all parties who have access to γ_i and λ_j secret share their product. The parties then compute the difference between the values shared as purported product $\gamma_i \lambda_j$ by securely computing their differences. If all differences come out to be 0, since at least one of the parties secret sharing the product is honest, all the remaining parties must also have correctly shared the secret. Hence, one of these CNF-shares can be taken as a valid secret sharing of $\lambda_i \gamma_j$. Whenever the difference is non-zero, both γ_i and λ_j are publicly revealed, and a trivial secret sharing of $\gamma_i \lambda_j$ is taken instead of the ones submitted by the parties. Finally, these shares are 'added' together to get a secret sharing of rs.

Multiplication functionality f_{Mult}

PUBLIC PARAMETERS: Input layer $\mathcal{L}_0$, output layer $\mathcal{L}_7$.
SECRET INPUTS: Each $\mathcal{P}_i^0$, $i \in [n]$ receives (r_i, s_i), where $(r_i)_{i \in [n]}$ and $(s_i)_{i \in [n]}$ are valid CNF secret sharing of r and s, respectively.

1. Each party $\mathcal{P}_i^0$, $i \in [n]$ sends (r_i, s_i) to f_{Mult}, who reconstructs r from $(r_i)_{i \in [n]}$ and s from $(s_i)_{i \in [n]}$. This is possible since at most t parties are corrupt and CNF secret sharing is t robust.
2. f_{Mult} samples $(u_1, \ldots, u_n)$ as a fresh CNF secret sharing of rs. For each $i \in [n]$, f_{Mult} delivers u_i to $\mathcal{P}_i^7$, $i \in [n]$ in the output layer.

Figure 8.12: Multiplication functionality f_{Mult}

The above protocol is clearly correct. The security of the protocol follows from the fact that, whenever all the parties submitting shares of $\gamma_i \lambda_j$ for some i, j are honest, the protocol never reaches the public reveal phase. A formal description of the protocol in the standard setting as constructed in [Mau06] is provided in Section 8.10.4. Our multiplication protocol is a porting of the above protocol to the layered setting. In the process, we face two main challenges.

Firstly, when the public check of equality between purported shares of $\gamma_j \cdot \lambda_{j'}$ provided by a pair of parties fails in step 2, γ_j and $\lambda_{j'}$ need to be revealed by every party (in the input layer) with access to these values. This is tackled exactly as in the Future Multicast protocol. Using Future Messaging, all parties in the input layer secret share each γ_i and λ_i they hold to the layer where the equality check is made; the parties in this layer then selectively reveal the additive shares for which any of the equality checks fails.

The second challenge is less straightforward to handle. If the protocol is naively ported to the layered model, VSS of $\gamma_j \cdot \lambda_{j'}$ will be available in two different layers: once in the layer that initiates the equality check, and then again in the final layer that computes the VSS of $r \cdot t$ as the sum of VSS of $\gamma_j \cdot \lambda_{j'}$ for all $j, j' \in [N]$. But then, the adversary can corrupt t parties in both these layers, and recover $\gamma_j \cdot \lambda_{j'}$ for each (j, j'). This is overcome as follows: For each j, j', consider the special party whose share of $\gamma_j \cdot \lambda_{j'}$ will be chosen in the final addition (if the all equality checks for $\gamma_j \cdot \lambda_{j'}$ succeeds). This party samples $(\delta_k)_{k\in[N]}$ as additive secret shares of $\gamma_j \lambda_{j'}$, and verifiable secret share each δ_k instead of directly secret sharing $\gamma_j \cdot \lambda_{j'}$. The equality check is now carried out to check if $\sum_k \delta_k$ shared by the special party equals the value shared by every other party. Finally, parties in the output layer receive a VSS of $\gamma_j \cdot \lambda_{j'}$ in which the ith share is $(\delta_k)_{k:i\in T_k}$. This avoids reuse of the same VSS in two layers. The protocol is presented in Fig. 8.13.

Π_{Mult}, an $(n, t, 7)$-layered protocol for realizing f_{Mult}

PUBLIC PARAMETERS: Input layer $\mathcal{L}_0$, output layer $\mathcal{L}_7$. A (t, n) robust perfectly secure secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$

INPUTS: Party $\mathcal{P}_i^0, i \in [n]$ has inputs (r_i, s_i), where $(r_i)_{i\in[n]}$ and $(s_i)_{i\in[n]}$ are CNF secret shares of r and s, respectively.

DEFINITIONS: Let $\mathcal{T} = \{T_1, \dots, T_N\} = \{T \subset [n] : |T| = n - t\}$.
For each $i \in [n]$, $s_i = (\gamma_j : i \in T_j, j \in [N])$
and $r_i = (\lambda_j : i \in T_j, j \in [N])$, where $\sum_{j\in[N]} \gamma_j = r$ and $\sum_{j\in[N]} \lambda_j = s$.

SUBROUTINES: Protocols Π_{FM}, Π_{VSS}.

1. For each $j, j' \in [N]$, concurrently execute protocol $\Pi_{j,j'}$ (described below).
2. For each $j, j' \in [N]$, let $(\delta_k^{j,j'})_{k:i\in T_k}$ be the output of party $\mathcal{P}_i^7, i \in [n]$ at the end of $\Pi_{j,j'}$. Then, $\mathcal{P}_i^7$ stores $(\delta_k)_{k:i\in T_k}$ as their shares of rs, where $\delta_k = \sum_{j,j'\in[N]} \delta_k^{j,j'}$.

SUB-PROTOCOL: $\Pi_{j,j'}$ for $j, j' \in [N]$

NOTATIONS: Let $I = T_j \cap T_{j'}$; i.e., $\mathcal{P}_i^0$ has both γ_j and $\lambda_{j'}$ if and only if $i \in I$. Fix $i^* \in I$. To avoid redundancy, denote γ_j by γ and $\lambda_{j'}$ by λ.

(i). Layer $\mathcal{L}_0$:

1. $\mathcal{P}_{i^*}^0$ samples $\delta_1, \dots, \delta_N$ as additive secret shares of $\gamma\lambda$. For each $k \in [N]$, execute Π_{VSS} with $\mathcal{P}_{i^*}^0$ as dealer with input δ_k and $\mathcal{L}_5$ as shareholders. For each $i \in I, i \neq i^*$, execute Π_{VSS} with $\mathcal{P}_i^0$ as dealer with input $\gamma\lambda$ and $\mathcal{L}_5$ as shareholders.
2. Each $\mathcal{P}_i^0, i \in T_j$ samples $(\gamma^{i,1}, \dots, \gamma^{i,n}) \leftarrow \mathsf{Sh}(\gamma)$. For each $k \in [n]$, execute Π_{FM} with $\mathcal{P}_i^0$ as sender with input $\gamma^{i,k}$ and $\mathcal{P}_k^6$ as receiver.
3. Similarly, Each $\mathcal{P}_i^0, i \in T_{j'}$ samples $(\lambda^{i,1}, \dots, \lambda^{i,n}) \leftarrow \mathsf{Sh}(\lambda)$. For each $k \in [n]$, execute Π_{FM} with $\mathcal{P}_i^0$ as sender with input $\lambda^{i,k}$ and $\mathcal{P}_k^6$ as receiver.

(ii). Layer $\mathcal{L}_5$:

1. For each $i \in I, i \neq i^*$, we will denote the value verifiably secret shared by $\mathcal{P}_i^0$ in step (i).1 by $(\gamma\lambda)^{(i)}$. If $\mathcal{P}_i^0$ is honest, $(\gamma\delta)^{(i)} = \gamma\lambda$. Parties in $\mathcal{L}_5$ locally compute that shares of $(\gamma\lambda)^{(i)} - \sum_{l=1}^{N} \delta_l$ and broadcast the shares of the sum to $\mathcal{L}_6$.
2. For each $k \in [N]$, $\mathcal{P}_l^5, l \in [n]$ reveals their share of δ_k to each party $\mathcal{P}_i^7, i \in T_k$ using Future Messaging Π_{FM}.

(iii). Layer $\mathcal{L}_6$:

1. Each $\mathcal{P}_k^6, k \in [n]$ recovers $\gamma^{i,k}$ as the output of Π_{FM} with $\mathcal{P}_i^0, i \in T_j$ as sender and $\mathcal{P}_k^6$ as receiver initiated in step (i).2. Similarly, $\mathcal{P}_k^6$ recovers $\lambda^{i,k}$ as the output of Π_{FM} with $\mathcal{P}_i^0, i \in T_{j'}$ as sender and $\mathcal{P}_k^6$ as receiver initiated in step (i).3.
2. If there exists $i \neq i^*$ such that $(\gamma\lambda)^{(i)} - \sum_{l=1}^{N} \delta_l \neq 0$, then $\mathcal{P}_k^6$ sends a complaint along with $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$ to all parties in the next layer.

(iv). Layer $\mathcal{L}_7$:

1. If at least $n - t$ complaints are received, $\mathcal{P}_l^7, l \in [n]$ recovers $\gamma^i = \mathsf{Rec}(\gamma^{i,1}, \dots, \gamma^{i,n})$ for each $i \in T_j$, and recovers $\lambda^i = \mathsf{Rec}(\lambda^{i,1}, \dots, \lambda^{i,n})$ for each $i \in T_{j'}$. Set γ as the unique value such that $\gamma = \gamma^i$ for at least $n - 2t$ distinct $i \in T_j$; similarly, set λ as the unique value such that $\lambda = \lambda^i$ for at least $n - 2t$ distinct $i \in T_{j'}$. Finally, $\mathcal{P}_l^7$ sets the share $(rs)_l^{j,j'}$ to be the trivial sharing of $\gamma\lambda$; i.e., $(rs)_l^{j,j'} = (\delta_k)_{k:l\in T_k}$, where $\delta_k = \gamma\lambda$, if $k = 1$ and $\delta_k = 0$ otherwise.
2. If less than t complaints were received, each party $\mathcal{P}_i^7, i \in [n]$ recovers the CNF share of δ_k (for each k such that $i \in T_k$) sent by $\mathcal{P}_l^5, l \in [n]$ using Π_{FM} in step (ii).2. Using these shares, $\mathcal{P}_i^7$ recovers δ_k and sets the share $(rs)_i^{j,j'}$ to be $(\delta_k)_{k:i\in T_k}$.

Figure 8.13: Layered protocol Π_{Mult}

We first establish properties of the subroutine $\Pi_{j,j'}$ that computes CNF shares of $\gamma_j \cdot \lambda_{j'}$ for each $j, j' \in [N]$. in the lemma below, proven in Section 8.10.5.

Lemma 8.5. *For any $j, j' \in [N]$, the following properties hold for $\Pi_{j,j'}$ when executed in the presence of an adversary $\mathcal{A}$:*

(a) There exists $(\delta_k)_{k\in[N]}$ such that $\sum_{k=1}^{N} \delta_k = \lambda_j \gamma_{j'}$, and each honest party $\mathcal{P}_i^7, i \in [N]$ outputs $(\delta_k)_{k:i\in T_k}$ at the end of $\Pi_{j,j'}$.

(b) Suppose parties $\mathcal{P}_i^0, i \in H$ are honest, then for any a, b, a', b',

$$\mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a, \lambda_{j'} = b) \equiv \mathrm{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a', \lambda_{j'} = b').$$

By statement (a) in Lemma 8.5, for each $j, j' \in [N]$, parties in the output layer correctly receive a CNF secret sharing of $\gamma_j \lambda_{j'}$. Hence, the output of the parties at the end of the protocol is a CNF secret sharing of $\sum_{j=1}^{N} \sum_{j'=1}^{N} \gamma_j \lambda_{j'} = rs$. By statement (b) in Lemma 8.5, if $\lambda_{j'}$ or γ_j is not known to the adversary, the output of $\Pi_{j,j'}$ does not reveal $\gamma_j \lambda_{j'}$. This ensures that the protocol is secure. The following theorem formally proves the security of the protocol.

Theorem 8.4. *The protocol Π_{Mult} (described in Fig. 8.13) is an $(n, t, 7)$-layered protocol realizing f_{Mult}, for $t < n/3$.*

Proof. We prove the security by constructing a simulator $\mathcal{S}$. In each layer $\mathcal{L}_l, 0 \le l \le 7$, let $\mathcal{P}_i^l, i \in C_l$, where $C_l \subset [n]$, be the set of corrupt parties. For each corrupt party $\mathcal{P}_i^0, i \in C_l$, $\mathcal{S}$ receives $(\gamma_k)_{k:i\in T_k}$ and $(\lambda_k)_{k:i\in T_k}$ from the environment. $\mathcal{S}$ chooses γ_k arbitrarily for each k such that γ_k is not provided by the functionality. The simulator then emulates all the honest parties and interacts with the adversary. For this, $\mathcal{S}$ is required to set the input to the honest parties in $\mathcal{L}_0$ (these are the only parties with inputs). For each $i \notin C_0$, $\mathcal{S}$ sets the shares of r and s of honest emulated party $\mathcal{P}_i^0$ to be $(\gamma_k)_{k:i\in T_k}$ and $(\lambda_k)_{k:i\in T_k}$, respectively.

For each pair j, j' such that γ_j and $\lambda_{j'}$ are known to some corrupt party, i.e., both $C_0 \cap T_j$ and $C_0 \cap T_{j'}$ are non-empty, the execution of $\Pi_{j,j'}$ is identical in both the simulation and a real execution. Furthermore, by statement (a) in Lemma 8.5, in each such $\Pi_{j,j'}$, all honest parties in $\mathcal{L}_7$ output valid CNF shares of $\gamma_j \lambda_{j'}$. The view of the adversary in the protocol and in the simulation differ only in that the view in each $\Pi_{j,j'}$ such that $T_j \cap C_0$ or $T_{j'} \cap C_0$ is empty. However, by statement (b) in Lemma 8.5, the view of the adversary is identically distributed irrespective of the value of γ_j and $\lambda_{j'}$. That these properties hold even when $\Pi_{j,j'}$ is executed in parallel for each j, j' can be argued using a hybrid argument along the lines of our previous proofs. This concludes the proof. □

Executing the above protocol in parallel realizes a parallel multiplication functionality.

8.4.4 Realizing MPC from Layered Multiplication and Addition

In this section, we construct a secure (n, t, d)-layered protocol for computing any given function f by evaluating an layered arithmetic circuit computing the function. Suppose each party $\mathcal{P}_i^0, i \in [n]$ in the input layer has $x_i \in \mathbb{F}$ as input, and each party in the output layer (specified later) wants to compute $f(x_1, \ldots, x_n)$. The secure computation of f proceeds in three phases: an input sharing phase, a circuit evaluation phase and an output reconstruction phase.

In the input sharing phase, each input client verifiably CNF secret shares their input. In the circuit evaluation phase, the layered protocol traverses the layered circuit that evaluates f and evaluates every gate in the circuit. Evaluating a gate amounts to securely computing a CNF secret sharing of the value on the output wire of each gate using the CNF secret sharing of the values on its input wires. Finally, in the output reconstruction phase, the secret sharing of the value on the output wire is revealed to the output clients.

We elaborate on the circuit emulation phase below. Let C be a layered arithmetic circuit over a field $\mathbb{F}$ with D layers that computes f. At the end of the input phase, the values on the input wires of all gates in layer one of C are simultaneously made available on the same layer of the protocol graph. In the circuit evaluation phase, the protocol keeps the invariant that, if a layer $i \in [D]$ of C is processed, then the values on all the output wires from layer i of C are simultaneously available of a specific layer of the protocol graph. The protocol can then process all gates in layer $i + 1$ of C preserving the invariant.

Recall that every gate in C is either a multiplication-by-constant gate, an addition gate or a multiplication gate. Given a CNF secret sharing of the value on the input wire(s) of a multiplication-by-constant gate or an addition gate, a secret sharing of the value on the output wire can be computed by locally processing the share. That is, the value on the output wire of the gate is available on the same layer (of the protocol graph) on which the values on the input wires have been secret shared. However, for a multiplication gate, computing a CNF secret sharing of the product of the values on the input wires using a t-secure protocol for multiplication consumes 7 layers. This poses a challenge when ensuring the invariant that the values on the output wires of all gates in a layer (of C) are made available on the same layer of the protocol graph. We get around this obstacle as follows: for a multiplication-by-constant or an addition gate G, after locally computing the secret sharing of the value on the output wire, we further compute a multiplication gate with value on the output wire of G as one input and the other input value being fixed to one (identity). This is achieved by using a trivial secret sharing of one as the other input and executing the layered protocol for multiplication which consumes $d = 7$ layers. Hence, we ensure the invariant we require.

A layered protocol computing any n-ary function f

PUBLIC PARAMETERS: A layered arithmetic circuit C over $\mathbb{F}$ with D levels that computes f. Output layer is $d = 7D + 6$.
SECRET INPUTS: Each input client $\mathcal{P}_i^0$, $i \in [n]$ has input x_i.
OUTPUTS: Each output client receives $f(x_1, \ldots, x_n)$.
SUBROUTINES: Protocols Π_{VSS}, and Π_{Mult} realizing functionalities f_{VSS}, and f_{Mult}, respectively.

Input sharing phase:

1. For each $i \in [n]$, execute in parallel Π_{VSS} with $\mathcal{P}_i^0$ as dealer with input x_i and $\mathcal{L}_5$ as shareholders.

Circuit evaluation phase:

Invariant: For $1 \leq i \leq D$, parties in $\mathcal{L}_{5+7(i-1)}$ hold verifiable secret shares of the values on the input wires of any gate in layer i of C. The parties in $\mathcal{L}_{5+7(i-1)}$ process any gate G_j in layer i of C as follows.

(i). G_i is a multiplication-by-constant or addition gate:

1. Parties in $\mathcal{L}_{5+7(i-1)}$ locally compute the CNF secret shares of the value on the output wire of G_j using CNF secret shares of the value(s) of the input wires of G_j.
2. Securely compute the product of the value on the output wire of G_j and one. This is achieved as follows: Execute Π_{Mult} with $\mathcal{L}_{5+7(i-1)}$ as input layer and $\mathcal{L}_{5+7i}$ as output layer. Parties in $\mathcal{L}_{5+7(i-1)}$ use the CNF secret sharing of the value on the output wire of G_j and a trivial CNF secret sharing of 1 as inputs to the computation.

(ii). G_i is a multiplication gate:

1. Execute Π_{Mult} with $\mathcal{L}_{5+7(i-1)}$ as input layer as $\mathcal{L}_{5+7i}$ as output layer. Here, each party in $\mathcal{L}_{5+7(i-1)}$ uses their respective CNF shares of the input wires as inputs. Each party in $\mathcal{L}_{5+7(i-1)}$ receives CNF shares of the product of these values at the end of Π_{Mult}.

Output reconstruction phase:

1. The parties in $\mathcal{L}_{5+7D}$ reveal the CNF shares of value on the output wire of the output gate to every output client in the next layer.
2. Each output client in $\mathcal{L}_{6+7D}$ recovers the value from the CNF shares.

Figure 8.14: Layered protocol computing any function f

Theorem 8.5. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$ and gates partitioned into layers $L_1, \ldots, L_D$. Then, for any $t < n/3$, the protocol in Fig. 8.14 is an $(n, t, 6 + 7D)$-layered MPC protocol for f.*

Proof. We argue the security of the protocol in $(f_{\mathsf{VSS}}^n, f_{\mathsf{Mult}}^n)$-hybrid model and appeal to sequential composition to prove security of the entire protocol. This is possible since the protocol essentially uses execution of (parallel) Π_{VSS} from $\mathcal{L}_0$ to $\mathcal{L}_5$, and then only parallel invocations of Π_{Mult} from $\mathcal{L}_{5+7(i-1)}$ to $\mathcal{L}_{5+7(i)}$ for each $1 \leq i \leq D$. Thus, the protocol amounts to sequential invocations of secure protocols implementing parallel VSS, addition and multiplication functionalities.

Consider the protocol in $(f_{\mathsf{VSS}}^n, f_{\mathsf{Mult}}^n)$-hybrid model obtained from the protocol in Fig. 8.14 by replacing executions of VSS and multiplication with f_{VSS}^n and f_{Mult}^n, respectively. First, all inputs are CNF secret shared to $\mathcal{L}_5$. The protocol ensures that the value on the output wire of every gate is freshly and randomly secret shared before being consumed by gates in the next layer of the circuit. Thus, for each $1 \leq i \leq D$, parties in layer $5+7(i-1)$ hold fresh CNF shares of the values on the input wires of gates $G \in L_i$. Proceeding in this manner, it can be seen that the parties in layer $5+7D$ hold the value on the output wire of the circuit. The parties in $\mathcal{L}_{6+7D}$ recover this value by reconstructing the shares provided by the parties in the previous layer. □

8.5 Efficient Layered MPC

In this section, we present an *efficient* implementation of perfectly t-secure layered MPC when $t < n/3$. To achieve this, we first build verifiable Shamir secret sharing. As in our previous implementation of MPC, the only non-trivial step in developing a protocol for general MPC after implementing VSS is that of achieving perfectly secure multiplication of two values that are secret shared. We build the multiplication protocol by porting a multiplication protocol of [CDN15, CDM00] from the standard setting to the layered setting. We present the formal constructions and proofs of their security in later sections. The security of the protocols is argued along the lines of our previous constructions, albeit, with slightly more complex proofs.

8.5.1 Verifiable Shamir Secret Sharing

In this section, we implement verifiable Shamir secret sharing in the layered setting with perfect t-security for $t < n/3$. This is realized by porting a protocol from the standard setting to the layered setting. We mostly face the same set of challenges that we encountered while implementing future multicast in the previous section. Recall that (t, n)-Shamir secret sharing of a secret s in a field $\mathbb{F}$ involves sampling a random polynomial $q(x)$ of degree at most t under the constraint $q(0) = s$ and setting the ith share to be $q(i)$. We consider an equivalent functionality $f_{\mathsf{ShamirVSS}}$ that allows a dealer to distribute the evaluation of a degree (at most) t polynomial on distinct non-zero points. A formal description of the parallel $f_{\mathsf{ShamirVSS}}$ functionality is given in Fig. 8.15.

Parallel $f_{\mathsf{ShamirVSS}}$ functionality for sharing polynomials of degree at most t.

PUBLIC PARAMETERS: Dealers $\mathsf{S}_1, \ldots, \mathsf{S}_n \in \mathcal{L}_0$, shareholders $\mathcal{P}_1^6, \ldots, \mathcal{P}_n^6$. A finite field $\mathbb{F}$ of size more than n.

1. $f_{\mathsf{ShamirVSS}}$ receives coefficients $c_{i,l} \in \mathbb{F}$ for $0 \leq l \leq t$ from each S_i, $i \in [n]$. $f_{\mathsf{ShamirVSS}}$ ignores any extra values sent by S_i and sets missing values as 0.
2. For each $i \in [n]$, define polynomial $q_i(x) = c_{i,t}x^t + \ldots + c_{i,1}x + c_{i,0}$, and send $q_i(j)$ to each $\mathcal{P}_j^6, j \in [n]$.

Figure 8.15: Parallel functionality $f_{\mathsf{ShamirVSS}}$

Implementing $f_{\mathsf{ShamirVSS}}$. The layered protocol realizing $f_{\mathsf{ShamirVSS}}$ is provided in Fig. 8.16. We sketch the outline and the intuition behind its construction.

The classic protocol for Shamir VSS in the standard setting proceeds as follows. Suppose the dealer wants to share a secret s from a field $\mathbb{F}$ such that $|\mathbb{F}| > n$ with t-security for $t < n/3$. The dealer samples a random bi-variate polynomial $S(x, y)$ of degree at most t in both the variables such that $S(0, 0) = s$, and transfers $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ to party P_i. If the polynomials were appropriately sampled, $f_i(j) = S(i, j) = g_j(i)$ for every i, j. Each pair of parties P_i, P_j check if $f_i(j) = g_j(i)$ and $f_j(i) = g_i(j)$; P_i raises a complaint by broadcasting $(i, j, f_i(j), g_i(j)$ if this check fails for P_j. The dealer addresses every valid complaint–a complaint of the form (i, j, u, v) such that $u \neq f_i(j)$ or $v \neq g_i(j)$–and broadcasts (f_i, g_i); otherwise, the dealer dismisses that complaint. This is followed by parties casting votes to accept or disqualify the dealer. P_i votes to accept the dealer if all the following conditions are met: dealer addressed one of every inconsistent mutual complaint–*i.e.*, a pair of complaints (i, j, u, v) and (j, i, u', v') such that $u \neq u'$ or $v \neq v'$; P_i itself did not issue a complaint; and for each broadcasted (f_j, g_j), $f_i(j) = g_j(i)$ and $g_i(j) = f_j(i)$. If the dealer receives less than $n - t$ votes, it is declared to be corrupt. Otherwise, each P_i updates (f_i, g_i) if it was broadcasted by the dealer and sets $f_i(0)$ as their share.

Using selective reveal in future messaging and checking equality using f_{Add} as done in future multicast, we can port the above protocol into the layered setting. The protocol obtained in this manner is used as sub-protocol Π in our final construction in Fig. 8.16. Interestingly, this construction by itself is not a layered protocol for verifiable secret sharing. However, Π guarantees the following: Let $H_1 \subseteq [n]$ such that $\mathcal{P}_i^1$ is honest iff $i \in H_1$; parties in $\mathcal{L}_5$ hold a secret sharing of a value $\hat{s}_i$ such that, all such $\hat{s}_i$ (there are at least $n - t$ of them) define a valid secret sharing of a value $\hat{s}$. Further, if the dealer is honest, $\hat{s} = s$ and $\hat{s}_i$ is the same as the value that the dealer transferred to $\mathcal{P}_i^1$. This is formally stated in Lemma 8.6.

Lemma 8.6. *The following properties hold for an execution of Π in the presence of a layered adversary $\mathcal{A}$:*

(a) Let $G \subseteq [n]$ such that $\mathcal{P}_i^1$ is honest if and only if $i \in H_1$. There exist polynomials $\hat{g}(x)$ and $\hat{g}_i(x), i \in H_1$, each of degree at most t, such that $\hat{g}_i(0) = \hat{g}(i)$ and α_i^k output by each honest party $\mathcal{P}_k^5$ coincides with $\hat{g}_i(k)$. Furthermore, if S is honest, $\hat{g}(x) = F(x, 0)$.

(b) If S is honest, for any $r, r' \in \mathbb{F}$,

$$\mathsf{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathsf{ADVR}_{\Pi,\mathcal{A}}(r').$$

An $(n, t, 6)$-layered protocol for realizing $f_{\mathsf{ShamirVSS}}$.

PUBLIC PARAMETERS: Dealer $\mathsf{S} \in \mathcal{L}_0$, shareholders $\mathcal{P}_1^6, \ldots, \mathcal{P}_n^6 \in \mathcal{L}_6$. A finite field $\mathbb{F}$, $|\mathbb{F}| > n$. A (t, n)-Shamir secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$.
SECRET INPUTS: Dealer holds a polynomial $q(x) = c_t x^t + \ldots + c_1 x + c_0$ of degree at most t over $\mathbb{F}$.
SUBROUTINES: Protocol Π_{FM} realizing f_{FM} functionality and Π_{Add} realizing f_{Add} functionality.

1. For each $0 \le l \le t$, concurrently execute protocol Π (described below) with c_l as input of S.
2. Each $\mathcal{P}_k^5$, $k \in [n]$ stores the output at the end of execution l of Π as $(\alpha_{l,i}^k)_{i\in[n]}$ for each $0 \le l \le t$. To each $\mathcal{P}_j^6$, $j \in [n]$, $\mathcal{P}_i^5$ sends $\gamma_{i,j}^k = \sum_{l=0}^t \alpha_{l,i}^k j^l$ for each $i \in [n]$.
3. Each $\mathcal{P}_j^6$ reconstructs $\gamma_{i,j} = \mathsf{Rec}(\gamma_{i,j}^1, \ldots, \gamma_{i,j}^n)$ for each $i \in [n]$. Finally, $\mathcal{P}_j^6$ outputs $\gamma_j = \mathsf{Rec}(\gamma_{1,j}, \ldots, \gamma_{n,j})$.

SUB-PROTOCOL: Π with S as dealer with input s.

(i). Layer 0.

1. S samples a random bivariate polynomial $F(x, y)$ of degree at most t in both variables conditioned on $F(0, 0) = s$. Let $f_i(x) = F(x, i)$ and $g_i(y) = F(i, y)$ for each $i \in [n]$. S privately sends (f_i, g_i) to $\mathcal{P}_i^1$.
2. For each $i, j \in [n]$, S samples $(F^1(i, j), \ldots, F^n(i, j)) \leftarrow \mathsf{Sh}(F(i, j))$. For each $k \in [n]$, S sends $F^k(i, j)$ to $\mathcal{P}_k^4$ using future messaging (Π_{FM}).
3. For each $i \in [n]$, let $f_i(x) = \sum_{l=0}^t \alpha_{i,l} x^l$ and $g_i(x) = \sum_{l=0}^t \beta_{i,l} y^l$. S samples $(\alpha_{i,l}^1, \ldots, \alpha_{i,l}^n) \leftarrow \mathsf{Sh}(\alpha_{i,l})$, and $(\beta_{i,l}^1, \ldots, \beta_{i,l}^n) \leftarrow \mathsf{Sh}(\beta_{i,l})$ for each $0 \le l \le t$. For each $i, k \in [n]$ and $0 \le l \le t$, S sends $\alpha_{i,l}^k$ and $\beta_{i,l}^k$ to $\mathcal{P}_k^5$ using future messaging.

(ii). Layer 1.

1. For each $i, j \in [n]$, execute Π_{Add} to compute $f_i(j) - g_j(i)$ and $g_i(j) - f_j(i)$ and broadcast the results to $\mathcal{L}_3$.
2. $\mathcal{P}_i^1$ samples $(f_i^1(j), \ldots, f_i^n(j)) \leftarrow \mathsf{Sh}(f_i(j))$, $(g_i^1(j), \ldots, g_i^n(j)) \leftarrow \mathsf{Sh}(g_i(j))$ for each $j \in [n]$, and sends both $f_i^k(j)$ and $g_i^k(j)$ to $\mathcal{P}_k^3$ and to $\mathcal{P}_k^4$ using future messaging.
3. $\mathcal{P}_i^1$ samples $(g_i^1(0), \ldots, g_i^n(0)) \leftarrow \mathsf{Sh}(g_i(0))$ and sends $g_i^k(0)$ to $\mathcal{P}_k^5$ using future messaging.

(iii). Layer 3.

1. Each $\mathcal{P}_k^3$, $k \in [n]$ recovers
   a) $F^k(i, j)$ as the output of future messaging initiated in step (i). 2. for each $i, j \in [n]$.
   b) For each $j \in [n]$, $\mathcal{P}_k^3$ recovers $f_i^k(j)$ and $g_i^k(j)$ as the output of future messaging initiated in step (ii).2 by $\mathcal{P}_i^1$, $i \in [n]$.
2. Using the output of Π_{Add}, $\mathcal{P}_k^3$ computes and broadcasts the set

$$S = \{\{i, j\} : f_i(j) - g_j(i) \neq 0 \text{ or } f_j(i) - g_i(j) \neq 0\},$$

   and $F^k(i, j)$, $F^k(j, i)$, $f_i^k(j)$, $g_i^k(j)$, $f_i^k(j)$ and $g_i^k(j)$ for each $\{i, j\} \in S$.

(iv). Layer 4.

1. For each $\{i, j\} \in S$ and $l \in [n]$, $F^l(i, j)$ was broadcasted by $\mathcal{P}_l^3$, $l \in [n]$. Each $\mathcal{P}_k^4$, $k \in [n]$ recovers $F(i, j) = \mathsf{Rec}(F^1(i, j), \ldots, F^n(i, j))$. Similarly, $\mathcal{P}_k^4$ recovers $F(j, i)$, $f_i(j)$, $g_i(j)$, $f_j(i)$, $g_j(i)$.

2. Each $\mathcal{P}_k^3$ recovers $\alpha_{i,l}^k$ and $\beta_{i,l}^k$ for each $0 \le l \le t$ as the output of future messaging initiated in step (i).3. with S as sender. $\mathcal{P}_k^3$ also recovers $f_i^k(j)$ and $g_i^k(j)$ for each $j \in [n]$ as the output of future messaging initiated in step (ii).2 with $\mathcal{P}_i^1$ as sender.
3. Define $B = \{i \in [n] : \exists j, \{i,j\} \in S$ and $(F(i,j) \neq g_i(j)$ or $F(j,i) \neq f_i(j))\}$. For each $i \in B$,
   a) $\mathcal{P}_k^4$ broadcasts $\alpha_{i,l}^k$ and $\beta_{i,l}^k$ for each $0 \le l \le k$.
   b) $\mathcal{P}_k^4$ broadcasts $f_j^k(i)$ and $g_j^k(i)$ for each $j \in [n]$.

(v). Layer 5.

1. For each $i \in B$ and $0 \le l \le t$, each $\mathcal{P}_k^5$ recovers $\alpha_{i,l} = \mathsf{Rec}(\alpha_{i,l}^1, \ldots, \alpha_{i,l}^n)$ and $\beta_{i,l} = \mathsf{Rec}(\beta_{i,l}^1, \ldots, \beta_{i,l}^n)$. Let $\hat{f}_i(x) = \sum_{l=1}^t \alpha_{i,l} x^l$ and $\hat{g}_i(y) = \sum_{l=1}^t \beta_{i,l} y^l$.
2. For each $i \in B$ and $j \in [n]$, each $\mathcal{P}_k^5$ recovers $f_j(i) = \mathsf{Rec}(f_j^1(i), \ldots, f_j^n(i))$ and $g_j(i) = \mathsf{Rec}(g_j^1(i), \ldots, g_j^n(i))$.
3. Define set $B' = \{j \in [n] : \exists i \in B, f_j(i) \neq \hat{g}_i(j)$ or $g_j(i) \neq \hat{f}_i(j)\}$.
   a) If $|B \cup B'| \ge t$, dealer is disqualified. Each $\mathcal{P}_k^5$ outputs $(\alpha_i^k)_{i \in [n]}$, where $\alpha_i^k = 0$ for all $i \in [n]$.
   b) Otherwise, for each $i \in [n] \setminus B$, $\mathcal{P}_k^5$ recovers $g_i^k(0)$ as the output of future messaging initiated in step (ii).3, and outputs $(\alpha_i^k)_{i \in [n]}$ such that $\alpha_i^k = g_i^k(0)$ if $i \in [n] \setminus B$ and $\hat{g}_i(0)$ otherwise.

Figure 8.16: Layered protocol for realizing $f_{\mathsf{ShamirVSS}}$

Using Π as a subroutine, verifiable secret sharing is achieved as follows (described in Fig. 8.16). Let $q(x) = c_0 + c_1 x + \ldots c_t x^t$ be the polynomial that the dealer wants to secret share. For each $0 \le l \le t$, dealer S executes Π with c_l as its input. When $\mathcal{P}_i^5$, $i \in H_5$ are the set of honest parties in $\mathcal{L}_5$. By Lemma 8.6, for each $0 \le l \le t$, there exist polynomials $\hat{g}_l(x)$ and $\{\hat{g}_{l,i}(x)\}_{i \in H_1}$ of degree at most t such that, $\hat{g}_{l,i}(0) = \hat{g}_l(i)$, and for all $k \in H_5$ and $l \in H_1$, $\mathcal{P}_5^k$ holds $\alpha_{l,i}^k = \hat{g}_{l,i}(k)$. Since $|H_5| \ge n - t$, each $\mathcal{P}_j^6, j \in [n]$ recovers

$$\gamma_{i,j} = \mathsf{Rec}(\alpha_{l,j}^1, \ldots, \alpha_{l,j}^n) = \sum_{l=0}^{t} \hat{g}_l(i) j^l, \forall i \in H_1$$

Hence,

$$\gamma_j = \mathsf{Rec}(\gamma_{1,j}, \ldots, \gamma_{n,j}) = \sum_{l=0}^{t} \hat{g}_l(0) j^l$$

Defining $\hat{q}(x) = \hat{g}_l(0) x^l$, we conclude that each $\mathcal{P}_j^6$ receives $\hat{q}(j)$ as required in verifiable Shamir secret sharing. When S is honest, by Lemma 8.6 (a), $\hat{g}_l(0) = c_l$ for each $0 \le l \le t$. Hence, $\hat{q}(x) = q(x)$.

We next argue that, when S is honest, the view of the adversary is identical irrespective of the value of $q(0)$. Assume that the guarantee in Lemma 8.6 (b) is preserved when Π is executed concurrently as in the protocol. Then, the view of the adversary till $\mathcal{L}_5$ are identically distributed in the protocol irrespective of the values of $(c_l)_{0 \le l \le t}$. Hence, the view of the adversary in the protocol only reveals $q(i)$ for $i \in C_6$, where $\mathcal{P}_i^6, i \in C_6$ are the set of corrupt parties in $\mathcal{L}_6$.

To formally argue the security of the protocol, we can build a simulator along the line of Theorem 8.2. Recall that, in the proof of Theorem 8.2, we used a hybrid argument to effectively extend the game based security guarantee of the weak multicast protocol $\Pi_{\text{weak-FMcast}}$ as stated in Lemma 8.4 (b) to parallel invocations as used in the protocol in Fig. 8.8. A similar argument can be used here.

In Fig. 8.16, the polynomial secret shared in $\mathcal{L}_6$ is exclusively determined by $\alpha^k_{l,i}$, for $i \in [n]$ and $0 \leq l \leq t$ stored by the honest parties $\mathcal{P}^5_k$. In other words, the dealer is committed to polynomial $\hat{g}_l(x), 0 \leq l \leq t$ (as described in Lemma 8.6) when all the honest parties in $\mathcal{L}_5$ finish receiving messages from their predecessors. Furthermore, by Lemma 8.6, when S is honest, view of the layered adversary is identically distributed irrespective of input of S in each invocation of Π. This ensures that, when the protocol for verifiable secret sharing is executed in parallel, the polynomial being secret shared by a corrupt dealer cannot be correlated with that shared by an honest dealer. In the following theorem, we state this stronger result: when n verifiable secret sharing protocols are executed in parallel with $\mathcal{P}^0_i, i \in [n]$ as dealer and $\mathcal{L}_6$ as shareholders, we realize a parallel VSS functionality with t-security. The parallel VSS functionality $f_{\text{ShamirVSS}}$ is formally described below.

Theorem 8.6. *The protocol described in [**?**] executed in parallel realizes $f_{\text{ShamirVSS}}$ with perfect t-security for $t < n/3$ by consuming 6 layers, and by communicating $O(n^6)$ field elements over the point-to-point channels and $O(n^4)$ field elements over the broadcast channels for each secret.*

Employing the layered protocol for VSS, we proceed to port the protocol for secure computation in [CDN15] to the layered setting. An important functionality we use extensively for this transformation is *resharing*, which allows parties in $\mathcal{L}_a$ with (a valid) secret sharing of a secret s to "handover" the secret to parties in $\mathcal{L}_b$, for any $b > a$, by providing a fresh secret sharing of s. Using parallel invocation of VSS, realizing resharing is straight forward: secret shares of uniformly random secrets $c_l, 1 \leq l \leq t$ are made available on the input layer. Then, the secret s is reshared by distributing $f(i)$ to shareholder i in the output layer; here $f(x) = s + \sum_{l=1}^{t} c_l x^l$. Distributing secret shares of a uniformly random secret is achieved by having $t+1$ parties in a previous layer secret share random secrets and the parties locally computing the shares of their sum (See functionality in Fig. 8.22 and its implementation in Fig. 8.23). The resharing functionality is formally defined in Fig. 8.24, and it is implemented as outlined above in Fig. 8.25.

8.5.2 Multiplication

The main challenge in realizing general MPC is securely implementing a multiplication protocol that computes a secret sharing of the product of two values using their shares. Following the outline of the MPC implementation in [CDN15], we first realize a simpler primitive, namely multiplication with helper, where the input clients hold secret sharing of a pair of values, and a special input client called the helper holds both values. This primitive allows the helper to verifiably secret share of the product of these values onto the output clients. The helper will be disqualified if the value secret shared is not the product.

8.5.2.1 Implementing multiplication with helper.

We realize this functionality by porting a modified version of the implementation of the same in standard setting as presented in [CDN15]. The protocol in the standard setting works as follows: When α, β are the values to be multiplied, helper samples polynomials $f(x)$ and $g(x)$ of degree at most t conditioned on $f(0) = \alpha$ and $g(0) = \beta$. It then computes $h(x) = f(x)g(x)$; clearly, $h(0) = \alpha\beta$. It then verifiably secret shares $(\alpha_l)_{l\in[t]}$, $(\beta_l)_{l\in[t]}$ and $(\gamma_l)_{0\leq l\leq 2t}$, where

$$f(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l \quad g(x) = \beta + \sum_{l=1}^{t} \beta_l x^l \quad h(x) = \sum_{l=0}^{2t} \gamma_l x^l$$

The parties now enter a verification phase in which $f(i), g(i)$ and $h(i)$ are revealed to P_i for each $i \in [n]$. P_i is to verify if $f(i)g(i) = h(i)$ and raise a complaint otherwise. For each complaint, $f(i), g(i)$ and $h(i)$ are publicly revealed; parties unanimously disqualify the helper if any of the complaint is valid. If all complaints turn out to be bogus, then $h(x)$ is verified to be $f(x)g(x)$ and $\gamma = \alpha\beta$. The parties now store the secret shares of γ as the shares of the product.

Our layered protocol follows the same logic with one notable difference. The helper in $\mathcal{L}_0$ secret shares the coefficients of $f(x), g(x)$ and $h(x)$ to $\mathcal{L}_6$ using the VSS protocol, with the exception of α and β. Recall that α and β are secret shared on $\mathcal{L}_0$; it is imperative to the correctness of the protocol that the secret shares of α and β provided to $\mathcal{L}_6$ are valid. But, this can be easily ensured by having α and β in $\mathcal{L}_0$ reshared to $\mathcal{L}_6$. In the standard setting, this is realized by "transferring" the secret sharing of α and β to the helper; resharing ensures the same guarantees. By taking appropriate linear combinations of the coefficients of the polynomials, parties in $\mathcal{L}_6$ then reveal $f(i), g(i)$ and $h(i)$ to each $\mathcal{P}_i^7, i \in [n]$. Each $\mathcal{P}_i^7$ raises a complaint if $f(i)g(i) \neq h(i)$ to $\mathcal{L}_8$. For each $i \in [n]$ with a complaint, parties in $\mathcal{L}_8$ selectively reveal $f(i), g(i)$ and $h(i)$ to all parties in $\mathcal{L}_9$. This is achieved by the trick we used in VSS as well as multicast and multiplication in the previous section. Finally, γ secret shared by the helper onto $\mathcal{L}_6$ is reshared to $\mathcal{L}_9$ and is used as the secret sharing of $\alpha\beta$ if the parties in $\mathcal{L}_8$ has not (unanimously) disqualified the helper.

When the helper is honest, throughout the protocol, the adversary only sees at most t shares of α, β, the evaluation of f, g and h on at most t points, and at most t shares of a sharing and resharing of γ. This ensures that the view of the adversary is identically distributed irrespective of the values of α and β. A corrupt helper is disqualified by the parties in $\mathcal{L}_8$ if and only if $h(x) \neq f(x)g(x)$. As we observed while analyzing the protocol for VSS, the sender commits to these coefficients by $\mathcal{L}_5$ as part of the VSS protocol. Hence, when this protocol is executed in parallel, a corrupt helper is unable to correlate the event of their getting disqualified with the secret sharing of the product achieved in another parallel execution with an honest helper. Thus, the protocol remains secure under parallel composition. The protocol is formally described in Fig. 8.29.

Theorem 8.7. *There is a layered protocol that realizes multiplication with helper functionality with perfect t-security for $t < n/3$.*

8.5.2.2 Multiplication.

We proceed to the main primitive required to implement MPC–secure processing of the multiplication gate. Suppose two values α, β are Shamir secret shared using polynomials $f(x)$ and $g(x)$. Since $f(x)g(x)$ is a polynomial of degree at most $2t$, given $f(i)g(i)$ for at least $2t+1$ distinct $i \in [n]$, there exists a linear transformation that computes $f(0)g(0) = \alpha\beta$. For each $i \in [n]$, suppose we execute the multiplication with helper protocol from the previous section to verifiably secret shares the product $f(i)g(i)$ with the help of the party holding $f(i)$ and $g(i)$. The protocol guarantees that the secret sharing of the product is accepted (and the helper is not disqualified) whenever the helper adheres to the protocol; whereas, if the helper secret shares a value other than the product then the helper is disqualified. Since at least $n - t$ parties are corrupt, we obtain the correct secret sharing of $f(i)g(i)$ for $n - t \geq 2t + 1$ distinct values of i, which can be locally transformed using the aforementioned linear transformation to obtain a secret sharing of $\alpha\beta$.

The above proposal has a clear flaw: to multiply $f(i)$ and $g(i)$ held by a helper, both these values need to be secret shared in the same layer. Hence, we need each $f(i)$ and $g(i)$ to be further secret shared onto the input layer. We refer to the 'data structure' where each share of a value is further verifiably secret shared as reinforced secret sharing (formally described in Fig. 8.27). The multiplication functionality takes *reinforced secret shares* of two values as input; to promote sequential processing of multiplication, we also ensure that the output of the functionality is a reinforced secret sharing of the product of the input values.

Multiplication functionality

PUBLIC PARAMETERS: Output layer $\mathcal{L}_{10}$.
INPUT: Reinforced secret sharing of two values a, b;
i.e., $[[a, f(x), (f_i(x))_{i\in[n]})]]_0$ and $[[b, g(x), (g_i(x))_{i\in[n]})]]_0$.

1. Input clients reveal shares $\langle a, f(x)\rangle_0$ and $\langle b, g(x)\rangle_0$ to the functionality who reconstructs a, b.
2. Functionality distributes a reinforced secret sharing of ab onto the output clients. That is, it distributes $\langle ab, h(x)\rangle_{10}$, where $h(x)$ is a random polynomial of degree at most t such that $h(0) = ab$; for each $i \in [n]$, it distributes $\langle h(i), h_i(x)\rangle_b$, where $h_i(x)$ is a random polynomial of degree at most t such that $h_i(0) = h(i)$.

Figure 8.17: Multiplication functionality

Implementing multiplication. The protocol takes valid reinforced secret sharing of two values as input. It then proceeds as outlined above by appropriately executing the multiplication with helper protocols and then linearly combining the evaluations on the product polynomial to obtain a *Shamir secret sharing* of the product of the inputs. It remains to convert the Shamir secret sharing to a reinforced secret sharing of the product; this is realized by executing a protocol for reinforced resharing implemented in Figure 8.27. The protocol for secure multiplication is formally presented in Figure 8.18.

A t-secure protocol for multiplication

PUBLIC PARAMETERS: Output layer $\mathcal{L}_d$.
INPUT: $[[\alpha, f(x), (f_i(x))_{i\in[n]})]]_0$ and $[[\beta, g(x), (g_i(x))_{i\in[n]})]]_0$.
SUBROUTINES: Π_{multh} and Π_{rreshare} implementing multiplication with helper and reinforced resharing, respectively.
SETUP: Random secret shares in appropriate layers as required to execute multiplication with helper and reinforced resharing.

(i). Product computation

1. For each $i \in [n]$, execute Π_{multh} for multiplication with $\mathcal{P}_i^0$ as helper to share $f(i)g(i)$ onto $\mathcal{L}_9$. The inputs to the protocol are $\langle f(i), f_i(x)\rangle_0$ and $\langle f(i), f_i(x)\rangle_0$. Additionally, the helper holds $f(i)$ and $g(i)$, the shares of $\mathcal{P}_i^0$ in $\langle \alpha, f(x)\rangle_0$ and $\langle \beta, g(x)\rangle_0$, respectively.
2. Let $G \subseteq [n]$ such that, for each $i \in G$, at the end Π_{multh} with $\mathcal{P}_i^0$ as helper, the parties stored $\langle f(i)g(i), h_i(x)\rangle_9$ as a valid secret sharing of $f(i)g(i)$ (instead of disqualifying the helper). There exist scalars $(r_i)_{i\in G}$ such that $\sum_{i\in G} r_i f(i)g(i) = f(0)g(0) = \alpha\beta$. Define

$$\langle \alpha\beta, h(x)\rangle_9 = \langle \sum_{i\in G} r_i f(i)g(i), \sum_{i\in G} r_i h_i(x)\rangle_9 = \sum_{i\in G} r_i \langle f(i)g(i), h_i(x)\rangle_9$$

(ii). Reinforced resharing of the product

1. Execute the t-secure protocol for reinforced resharing with $\langle \alpha\beta, h(x)\rangle_9$ as input from $\mathcal{L}_9$ and $\mathcal{L}_{10}$ as the output layer, and random secret shares distributed onto $\mathcal{L}_9$ as setup. That is, execute $[[\alpha\beta, h'(x), ((h_i')(x))_{i\in[n]}]]_{10} \leftarrow \langle \alpha\beta, h(x)\rangle_9$.
2. The parties in $\mathcal{L}_4$ store their respective shares of $[[\alpha\beta, h'(x), ((h_i')(x))_{i\in[n]}]]_{10}$.

Figure 8.18: A t-secure protocol for multiplication

The protocol inherits security from the security of protocols implementing (parallel) multiplication with helper and reinforced resharing since the protocol exclusively uses these protocols in parallel. Indeed, the protocol remains secure under parallel composition because both the subroutines remain secure under parallel composition.

Theorem 8.8. *There is a layered protocol that realizes multiplication functionality with perfect t-security for $t < n/3$.*

8.5.3 MPC

In this section, we construct an efficient t-secure protocol for securely computing any given function f by evaluating a layered arithmetic circuit C computing the function. Suppose each party $\mathcal{P}_i^0$, $i \in [n]$ in the input layer has $z_i \in \mathbb{F}$ as input, and each party in the output layer (specified later) wants to compute $f(z_1, \ldots, z_n)$. Similar to our CNF secret sharing based construction, the secure computation of f proceeds in three phases: an input sharing phase, a circuit evaluation phase and an output reconstruction phase.

In the input sharing phase, each input client secret shares their input using reinforced secret sharing. In the circuit evaluation phase, the protocol keeps the invariant that, if a layer i of C is processed, then the values on all the output wires outgoing from layer i of C are simultaneously available of a specific layer of the protocol graph. Given a reinforced secret sharing of the value on the input wire(s) of a multiplication-by-constant gate or an addition gate, a secret sharing of the value on the output wire can be computed by locally processing the share. However, for a multiplication gate, computing a Shamir secret sharing of the product of the values on the input wires using a t-secure protocol for multiplication consumes 10 layers. Hence, we once again face the challenge of ensuring the invariant that the values on the output wires of all gates in a layer (of C) are made available on the same layer of the protocol graph. We get around this obstacle the same way we did in our previous construction: for a multiplication-by-constant or an addition gate G, after locally computing the reinforced secret sharing of the value on the output wire, we further compute a multiplication gate with value on the output wire of G as one output and the other value being fixed to one. This is achieved by taking a trivial secret sharing of one as the other input and executing the t-secure protocol for multiplication which consumes 10 layers. In this manner, we preserve the invariant we require. The protocol is formally described in Fig. 8.19.

In the protocol for secure computation of f, we will use the protocol for multiplication presented in Fig. 8.18 and the protocol for reinforced resharing presented in Fig. 8.27. Both these subroutines use several random secret sharing and random secret sharing with various owners as setup. We generate the setup needed to execute these protocols using the random secret sharing protocol presented in Fig. 8.23. We will argue that the protocol remains secure when the setup is computed in parallel.

Theorem 8.9. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $t < n/3$, there is an $(n, t, 8+10D)$-layered MPC protocol for f in which the communication consists of $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^7)$ field elements over the broadcast channels.*

Efficient layered protocol for computing a function f

PUBLIC PARAMETERS: A layered arithmetic circuit C over $\mathbb{F}$ with D levels that computes the desired function f. Output layer $10D + 7$.
SECRET INPUTS: Each input client $\mathcal{P}_i^0$, $i \in [n]$ has input z_i.
SUBROUTINES: protocols implementing VSS (Figure 8.16), random secret sharing (Figure 8.23),
reinforced resharing (Figure 8.27) and multiplication (Figure 8.18).

Input sharing stage:

1. For each $i \in [n]$, execute verifiable secret sharing with $\mathcal{P}_i^0$ as dealer with input z_i and $\mathcal{L}_7$ as shareholders to distribute $\langle z_i, p_i(x)\rangle_6$.
2. For each $i \in [n]$, execute reinforced resharing protocol from $\mathcal{L}_6$ to $\mathcal{L}_7$ to realize $[[z_i, f_i(x), (f_{i,j}(x))_{j\in[n]}]]_7 \leftarrow \langle z_i, p_i(x)\rangle_6$.

Circuit evaluation stage:

Invariant: For $1 \le i \le D$, parties in $\mathcal{L}_{7+10(i-1)}$ hold reinforced secret shares of the values on the input wires of any gate in layer i of C. The parties in $\mathcal{L}_{7+10(i-1)}$ process any gate G_j in layer i of C as follows:

(i). G_j is a multiplication-by-constant or addition gate:

1. Parties in $\mathcal{L}_{7+10(i-1)}$ locally compute the reinforced secret sharing $[[o_j, g_j(x), (g_{j,k})_{k\in[n]}]]_{7+10(i-1)}$ of the value on G_j's output wire.
2. Securely compute the product of value on the output wire of G_j with identity. This is achieved as follows: Execute the t-secure protocol for multiplication in with $\mathcal{L}_{7+10(i-1)}$ as input layer and $\mathcal{L}_{7+10i}$ as output layer. Parties in $\mathcal{L}_{7+10(i-1)}$ use $[[o_j, g_j(x), (g_{j,k})_{k\in[n]}]]_{7+10(i-1)}$ and a trivial reinforced secret share of 1 given by $[[1, 1(x), (1(x))_{i\in[n]}]]_{7+10(i-1)}$ as inputs to the computation. Here, $1(x)$ is the constant polynomial that evaluates to 1 on every point.

(iii). G_i is a multiplication gate:

1. Execute the protocol for multiplication with $\mathcal{L}_{7+10(i-1)}$ input layer and $\mathcal{L}_{7+10i}$ as output layer. Parties in $\mathcal{L}_{7+10(i-1)}$ use the reinforced secret shares of the values on the input wires as inputs for this computation.

Output reconstruction phase:

1. The parties in $\mathcal{L}_{7+10D}$ reveal the reinforced secret shares of the output wire of C to every output client in the next layer.
2. Each output client in $\mathcal{L}_{8+10D}$ recovers the value on the output wire from the reinforced secret shares.

Implementing setup:

1. Subroutines for reinforced resharing and for multiplication used setup at various levels. In order to obtain random secret sharing on $\mathcal{L}_a$, execute the t-secure protocol in Figure 8.23. This protocol uses verifiable secret sharing to be invoked by parties $\mathcal{P}_i^{a-7}$, $i \in [t+1]$. But, random secret sharing is invoked for the first time in $\mathcal{L}_7$.

Figure 8.19: Efficient layered protocol for computing a function f

8.6 Computational Efficient Layered MPC for $t < n/2$

We introduce a computationally-secure layered MPC protocol with guaranteed output delivery, based on (non-interactive) equivocal linearly homomorphic commitments. We give a high-level overview and defer details to [DGLZ23].

Future Messaging. The primitive is achieved similarly as its perfectly-secure counterpart, but to tolerate $t < n/2$ corruptions, we cannot rely on plain error correction. Instead, parties broadcast commitments to (coefficients of) the polynomials used to share their values to the future layers. Every time a party wishes to re-share an intermediate value, they re-use the commitment to the constant coefficient, thereby ensuring that the proper value is being re-shared.

Distributed Commitments. This primitive (also referred to as *weak* secret sharing [GIKR01]) allows a dealer $\mathsf{D} \in \mathcal{L}_c$ to commit to a value s towards a future layer $\mathcal{L}_{c'}$, and later open the original value towards another further layer $\mathcal{L}_{c''}$. If D is honest, the opened value is s, and no information about s is revealed before the opening phase. Moreover, even if D is corrupted, the commit phase uniquely determines value s', such that the opening phase can only output s' or $\perp$.

The dealer $\mathsf{D} \in \mathcal{L}_c$ samples random degree-t polynomials $f(x), r(x)$, such that $f(0) = s$, computes a commitment to each coefficient in f using the coefficients in r as randomness and broadcasts these commitments to the future layers. The dealer then sends the evaluation points $(s(i), r(i))$ using future messaging to party $P_i^{c'}$. To reconstruct, layer $\mathcal{L}_{c'}$ broadcast these pairs to the future layers, and each party $P_i^{c''} \in \mathcal{L}_{c''}$ checks for each received pair whether it is consistent with the corresponding commitment. If there are more than t consistent pairs, interpolate a degree-t polynomial $f'(x)$ and output $f'(0)$. Otherwise, output $\perp$.

Remark 8.3. *We can achieve a distributed commitment that allows to commit to the same value towards separate layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{c''}$, such that even if P_d^c is corrupted, there exists a unique value s' such that the value that is opened by either layer is s' or $\perp$: let the dealer P_d^c execute the above protocol towards layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{c''}$ with polynomials $f(x)$ and $f'(x)$ such that $f(0) = s = f'(0)$, but using the same commitment for the constant coefficient.*

Verifiable Secret Sharing. For VSS, the dealer $\mathsf{D} \in \mathcal{L}_c$ with input s, samples random degree-t polynomial $f(x)$ with $f(0) = s$, and (duplicate) commits to each coefficient of f towards layers $\mathcal{L}_{c_1}$ and $\mathcal{L}_{c'}$ using the distributed commitment. This results in a matrix $\mathbf{M} = [\mathsf{cm}_{i,j}]_{0 \le i,j \le t}$ of public commitments, where $\mathsf{cm}_{i,j}$ is a commitment to the j-th coefficient of the polynomial used to share f_i: by linearity of the commitment, parties implicitly hold commitments to each evaluation $f(i)$. Using future messaging, the dealer D sends $s_i = f(i)$ and its opening information to $P_i^{c_1}$. Party $P_i^{c_1}$ can check that the received information is consistent with the published commitments, and broadcast to future layers a

complaint if the check does not succeed. If the check succeeds, $P_i^{c_1}$ commits to s_i towards layer $\mathcal{L}_{c'}$; to ensure that $P_i^{c_1}$ commits to the value they received from P_c^d, the party uses the commitment to the constant term that is implicit from the published commitments in $\mathbf{M}$. If a complaint was raised by $P_i^{c_1}$, parties in layer $\mathcal{L}_{c'}$ publicly open s_i (and if the opened value is $\perp$, the dealer is disqualified). To reconstruct: for each index i corresponding to a party that did not complain, parties jointly reconstruct s_i. The final layer $\mathcal{L}_{c''}$ then uses any $t+1$ reconstructed shares to interpolate the secret. Moreover, as in Remark 2, by having both D and parties in layer $\mathcal{L}_{c_1}$ duplicate distribute commitments of s_i for all $i \in [1, n]$ towards layers $\mathcal{L}_{c'}$ and $\mathcal{L}_{\tilde{c}}$ for some $\tilde{c} \geq c'$, one guarantees that if D is not disqualified, then both $\mathcal{L}_{c'}$ and $\mathcal{L}_{\tilde{c}}$ hold sharings of the *same* value.

2-Level Verifiable Secret Sharing. To simplify the description of the MPC protocol, it is helpful that each party holds as part of their state a Shamir share of each wire value. For that, we modify the VSS as follows: the dealer D uses the above duplicate VSS to distribute shares of coefficients of f towards layers $\mathcal{L}_{c_1}$ and $\mathcal{L}_{c'}$, where $f(x)$ is a random degree-t polynomial with $f(0) = s$. Then, each party in $\mathcal{L}_{c_1}$ (privately) reconstructs towards party $P_i^{c'}$ the value $s_i = f(i)$. Notice that layer $\mathcal{L}_{c'}$ also holds sharings of all values $f(j)$ for $j \in [1, n]$ thanks to linearity of our VSS. This version of VSS can also be similarly duplicated.

Circuit Evaluation. Input parties use 2-level VSS to distribute their inputs towards a future layer. For each computation gate we maintain the invariant that layer $\mathcal{L}_c$ holds sharings (resulting from the 2-level VSS) of the input wire values x and y, and some future layer $\mathcal{L}_{c'}$ for $c' \geq c + 6$ holds a sharing of the output wire value z. Addition gates are processed locally, exploiting the linearity of 2-level-VSS. Multiplication gates are processed by adapting a well-known protocol of Cramer et al. [CDD+99]: each party $P_i^c \in \mathcal{L}_c$ holds (as part of their 2-level VSS states to x and y) Shamir shares x_i and y_i of each value, and computes a 2-level VSS for x_i, y_i (but using the already known sharing to the constant coefficient of the used polynomial) and a fresh 2-level VSS for $z_i = x_i \cdot y_i$ towards a future layer. Finally, each party carries out a distributed *multiplication proof* (adapted from [CDD+99]) to prove that indeed $z_i = x_i \cdot y_i$: if this proof fails, parties jointly reconstruct (and adopt a standard sharing of) x_i and y_i to continue the computation.

Theorem 8.10. *Let f be an n-party functionality computed by a layered arithmetic circuit C over a finite field $\mathbb{F}$, with D levels and M gates. Then, for any $t < n/2$, there is an $(n, t, 4+6D)$-layered MPC protocol for f assuming non-interactive linearly-homomorphic equivocal commitments. The communication complexity is $M \cdot O(n^9)$ field elements over the point-to-point channels and $M \cdot O(n^5)$ field elements + $M \cdot O(n^{10} \cdot \lambda)$ bits over the broadcast channels, where λ is the security parameter.*

8.7 Efficient CNF-based MPC with Computational Security

We can use standard techniques to improve the efficiency of our constructions if the security guarantee is relaxed from perfect to computational while maintaining perfect

correctness but achieving only static security. Verifiably secret sharing N distinct values using the scheme we presented in Figure 8.16 requires communicating $N \cdot O(n^6)$ field elements in total. We show a simple scheme that realizes secret sharing of random secrets with a preprocessing cost of $2^{O(n)}$ bits but uses no communication for subsequent sharing of unlimited random secrets.

Let f be a pseudorandom function that takes a λ bit long seed. When $\mathcal{T} = \{T \subset [n] : |T| = n-t\}$, the dealer uses future multicast to transfer a λ-bit seed s_T to parties $(\mathcal{P}_i^6)_{i \in T}$. This requires $2^{O(n)} \cdot \lambda$ communication in total. Next, to sample the secret share of the ith random secret r_i, each party computes $\delta_T = f_{s_T}(i)$ for each $T : i \in T$ and stores $(\delta_T)_{T:i \in T}$ as their CNF share. The dealer knows r_i since it has access to $(s_T)_{T \in \mathcal{T}}$, so it can broadcast a single value $s_i - r_i$ to $\mathcal{L}_6$ using $4n+1$ broadcasts, allowing the parties obtain shares of an arbitrary value s_i from their shares of r_i. In this protocol, a computationally bounded adversary only learns $s - r$, which is independent of s. Even when the dealer is corrupt, since r is correctly secret shared, the shares held by the parties remain valid at the end of this transformation. Asymptotically, this protocol making black-box use of a PRG uses $O(\ell)$ bits of communication per secret.

Theorem 8.11. *There exists a* $(n, t, 5)$*-layered protocol with* computational t*-security against a static malicious adversary and perfect* t*-correctness making black-box use of a PRG with seed length* λ*, that verifiable secret shares* ℓ *bits with* $\lambda \cdot 2^{O(n)} + O(n\ell)$ *bits of communication.*

We now obtain an asymptotically efficient layered protocol for general secure computation with computational t-security and perfect correctness for $t < n/3$. In our construction of Figure 8.19, the $O(n^3)$ invocations of VSS per gate dominate the communication cost. We tweak this protocol by realizing Shamir secret sharing by first distributing CNF shares using the VSS protocol of Theorem 8.11 and applying share conversion [CDI05] to obtain Shamir secret shares. The resulting protocol computes each gate using $O(n^5)$ bits of amortized communication.

Theorem 8.12. *Let* f *be an* n*-party functionality computed by a layered arithmetic circuit* C *over a finite field, with* D *levels and* M *gates. Then, for any* $t < n/3$*, there is an* $(n, t, O(D))$*-layered MPC protocol for* f *with* computational t*-security against a static malicious adversary and perfect correctness making black-box use PRG with seed length* λ*, and using* $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ *bits of communication.*

Given a Boolean circuit C computing a function f, a constant depth arithmetic circuit with $O(c)$ gates over a field of characteristic 2 that computes a garbled circuit of C can be constructed as in [DI05]. Using the protocol constructed in the above theorem to evaluate this circuit, we get the following result:

Corollary 8.2. *Let* f *be an* n*-party functionality computed by a Boolean circuit* C *with* M *gates. Then, for any* $t < n/3$*, there is an* $(n, t, O(1))$*-layered MPC protocol for* f *with* computational t*-security against a static malicious adversary and perfect correctness making black-box use PRG with seed length* λ*, and using* $\lambda \cdot 2^{O(n)} + O(n^5 \cdot M)$ *bits of communication.*

8.8 Layered Broadcast

The definition of layered MPC (Definition 8.1) assumes layer-to-layer broadcast. This turns out to be a necessary since deterministic broadcast is impossible in the layered setting even if only considering a fail-stop adversary. [Gar94] also explored the relation between an adversarial mobility and the ability to reach agreement in his work on Byzantine Agreement with Mobile Faults (MBA). They identified a mobile fault setting $\mathrm{MF}(\frac{t}{n-1}, \rho)$ parameterized by the adversarial threshold t of all n parties and "roaming speed" ρ. Indeed, the setting of $\mathrm{MF}(\cdot, 1)$ (full "roaming speed") is enough to render BA impossible. We now cast the result of [Gar94] in the setting of layered MPC.

Theorem 8.13. *Deterministic perfect broadcast for layered MPC is possible iff* $t = 0$.

The proof sketch below follows the technique of [FL81]. They use equivalence classes of executions to argue the lower bound of $t+1$ on the round complexity of unauthenticated broadcast. This was later extended to the authenticated setting in the seminal work of [DS83].

Proof Sketch. Assume the existence of a protocol Π for broadcast over a layered graph consuming d layers *i.e.* with output being obtained in layer $d+1$ and assume for simplicity that $n = 4$. Furthermore, we assume that a faulty party merely drops a subset of its outgoing messages and will otherwise follow the protocol.

The idea is to build a chain of "equivalent" executions where each execution has at most one faulty party in each layer and where (1) the first execution in the chain has 0 as the unique decision value (2) the last execution has 1 as the unique decision value (3) any two consecutive executions in the chain are indistinguishable to some honest party in the last layer.

Let the initial execution be the case where everyone are honest and the dealer has input 0. We say that a party is "muted" if it has no outgoing messages. Switching the input of a muted party has no effect on protocol execution. The crux of the proof is to show a sequence of equivalent executions that mutes a party in $\mathcal{S}^0$ and flips its input to 1 and then un-mutes it without violating requirement (3).

8.9 Basic Primitives

8.9.1 Details omitted from Section 8.3.1.2

We continue the discussion on parallel composition of f_{FM} from Section 8.3.1.2 in the remark below.

Remark 8.4. *When our protocol implementing f_{FM} is composed in parallel, the resulting functionality is not the natural parallel composition of f_{FM} which takes the input from each sender to each receiver and delivers them. In fact, this functionality is impossible to realize even in the trivial case of messaging from one layer to the very next using the provided secure communication link. As an example, suppose communication from $\mathsf{S}_1 \in \mathcal{L}_0$ to $\mathsf{R}_1 \in \mathcal{L}_1$ and from $\mathsf{S}_2 \in \mathcal{L}_0$ to $\mathsf{R}_2 \in \mathcal{L}_1$ are composed in parallel. Now, a rushing adversary corrupting S_1 and R_2 can collect the message from S_2 to R_2 and set this as the message from S_1 to R_1. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. Interestingly, this limitation persists when parallely composing our protocol for realizing f_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ (even for $d > 1$) with t-security for $t < n/3$. We demonstrate this for $d = 2$. In this case, the future messaging protocol proceeds as follows: The sender in $\mathcal{L}_0$ secret shares the message among the parties in $\mathcal{L}_1$ using a (t, n)-robust secret sharing scheme. Parties forward the shares they received from the sender to the receiver in $\mathcal{L}_2$ who reconstructs the secret from the shares. Since at least $n - t$ shares are forwarded unchanged by the honest parties in $\mathcal{L}_1$ the receiver correctly recovers the message; this follows from the secret sharing scheme being t-robust. However, a corrupt sender can distribute invalid shares to parties in $\mathcal{L}_1$. This allows the corrupt parties (colluding with the sender) to divulge shares that (together with shares forwarded by the honest parties) reconstruct to a value of their choosing, which they could decide on after collecting the views of corrupt parties in the next layer by rushing.*

8.9.2 Proving Lemma 8.2

Lemma 8.2 (Layered protocol for f^n_{FM})**.** *Let* $(\mathsf{Sh}, \mathsf{Rec})$ *be a robust* (t, n) *secret-sharing scheme Definition 5.1, the* (n, t, d)*-layered protocol in Figure 8.4 realizes the functionality* f^n_{FM} *in Figure 8.3 with perfect security for* $t < n/3$.

Proof. We prove the security of the protocol by presenting a simulator $\mathcal{S}$ which, given oracle access to f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$, simulates the view of the adversary interacting with the protocol in Figure 8.4.

The simulator $\mathcal{S}$ works in two phases: In the first phase, $\mathcal{S}$ emulates f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. Suppose the adversary corrupts $\mathcal{I} = \{\mathcal{P}^\ell_k, k \in I\} \subset \mathcal{L}_\ell$. In the protocol, as leakage from f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$, the adversary expects $(s_{(i,j),k})_{k\in I}$ among the shares $(s_{(i,j),k})_{k\in[n]}$ sampled according to $\mathsf{Sh}(m_{(i,j)})$ for each sender S_i and receiver R_j. These messages are simulated (without knowing $m_{(i,j)}$) by sampling them according to $\mathsf{Sh}(m)$ for an arbitrary member of M. On receiving this leakage, the adversary sends all messages that each corrupt S_i intends to send over f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$. The simulator now has all the values needed to simulate the output of f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ to all corrupt parties.

In the second phase, simulator emulates f^n_{FM} from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. For this, it first invokes f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_d$ to receive the message $m_{(i,j)}$ from every honest S_i to every corrupt R_j. The adversary expects $(s_{(i,j),k})_{k\in I}$ received during f^n_{FM} from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ in the first phase of the simulation, and $(s_{(i,j),k})_{k\notin I}$ it will receive in this phase to be jointly distributed according to $\mathsf{Sh}(m_{(i,j)})$. Recall that, $\mathcal{S}$ sampled $(s_{i,j),k})_{k\in I}$ according to $\mathsf{Sh}(m)$ in the previous

phase. But, since $|I| \leq t$, the simulator can patch $(s_{(i,j),k})_{k \notin I}$ such that $(s_{(i,j),k})_{k \in [n]}$ is distributed according to $\mathsf{Sh}(m_{(i,j)})$; this follows from t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$. After resampling $(s_{(i,j),k})_{k \notin I}$, the simulator can safely provide the leakage from f_{FM}^n from $\mathcal{L}_\ell$ to $\mathcal{L}_d$. At this point, the adversary reveals the set of all messages that each corrupt $\mathcal{P}_k^\ell$ intends to send over f_{FM}^n from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.

The simulator now holds all the messages that are needed to compute the messages from every corrupt sender to every receiver (corrupt or honest). Hence, it can extract the appropriate inputs that the corrupt parties need to input to f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_d$. It remains to argue that, for any honest S_i and honest R_j, the output of R_j in the protocol coincides with the input of S_i. This follows from t-robustness of $(\mathsf{Sh}, \mathsf{Rec})$ since each $s_{(i,j),k}$ sent by S_i via an honest $\mathcal{P}_k^\ell$ (there are at least $n - t$ of them) correctly reaches R_j.

Formally, consider any layered adversary $\mathcal{A}$ that non-adaptively corrupts a set of parties $\mathcal{I} \subset \mathcal{L}_\ell$ such that $|\mathcal{I}| \leq t$ and interacts with parties executing Π_{FM}^n. We will show that, for any input $m_{(i,j)} \in M, i, j \in [n]$,

$$\mathsf{EXEC}_{f_{\mathsf{FM}}^n, \mathcal{S}, \mathcal{I}}(m_{(i,j)}, i, j \in [n]) \equiv \mathsf{EXEC}_{\Pi_{\mathsf{FM}}^n, \mathcal{A}, \mathcal{I}}(m_{(i,j)}, i, j \in [n]). \tag{8.1}$$

Here, we used the notation from [Can00] to denote the joint distribution of the honest parties' outputs and the output of the adversary. When input to Π is $(x_1, \ldots, x_n)$, the random variable $\mathsf{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)$ is the vector

$$\begin{aligned} \mathsf{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n) = (&\mathsf{ADVR}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n), \\ &\mathsf{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)_1, \\ &\ldots \\ &\mathsf{EXEC}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)_n) \end{aligned}$$

where $\mathsf{ADVR}_{\Pi, \mathcal{A}, \mathcal{I}}(m'_1, \ldots, m'_n)$ is the output of $\mathcal{A}$, and, $\mathsf{EXEC}_{\Pi, \mathcal{A}, \mathcal{P}}(x_1, \ldots, x_n)_i$ is the output of party $\mathcal{P}_i^d$ for each $i \in [n]$ at the end of the interaction between Π invoked with input $(x_1, \ldots, x_n)$ and $\mathcal{A}$.

Simulator $\mathcal{S}$ works as follows:

Simulator $\mathcal{S}$ for the protocol in Figure 8.4

PUBLIC PARAMETERS: Senders $\mathsf{S}_1, \dots, \mathsf{S}_n \in \mathcal{L}_0$, receivers $\mathsf{R}_1, \dots, \mathsf{R}_n \in \mathcal{L}_d$ where $d > 0$.
ADDITIONAL INPUTS TO THE SIMULATOR: Set of corrupt parties $\mathcal{I}$, and oracle access to adversary $\mathcal{A}$.

1. Emulating f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_\ell$.
   a) For each honest S_i, $\mathcal{S}$ samples $(s_{(i,j),1}, \dots, s_{(i,j),n})$ according to $\mathsf{Sh}(m)$ for every $\mathsf{R}_j, j \in [n]$. Here, m is an arbitrary member of M. For each corrupt $\mathcal{P}_k^\ell$, $\mathcal{S}$ leaks $(s_{(i,1),k}, \dots, s_{(i,n),k})$ to $\mathcal{A}$ as message that S_i intends to send to $\mathcal{P}_k^\ell$ over f_{FM}^n.
   b) For each corrupt S_i and each $\mathcal{P}_k^\ell$, $\mathcal{S}$ receives $(s_{(i,1),k}, \dots, s_{(i,n),k})$ from $\mathcal{A}$ as the message that S_i intends to send to party $\mathcal{P}_k^\ell$ over f_{FM}^n.
   c) Finally, for each corrupt $\mathcal{P}_k^\ell$, $\mathcal{S}$ delivers $(s_{(i,1),k}, \dots, s_{(i,n),k})$ to $\mathcal{A}$ as message from $\mathsf{S}_i, i \in [n]$ to $\mathcal{P}_k^\ell$ over f_{FM}^n.
2. Emulating f_{FM}^n from $\mathcal{L}_\ell$ to $\mathcal{L}_d$.
   a) For each honest S_i and corrupt R_j, $\mathcal{S}$ receives the message $m_{(i,j)}$ that S_i intends to send to R_j as leakage from f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_d$.
   b) Let $I \subset [n]$ such that $\mathcal{P}_k^\ell$ is corrupt if and only if $k \in I$. For each honest S_i and corrupt R_j, resample $(s_{(i,j),k})_{k \notin I}$ such that $(s_{(i,j),k})_{k \in [n]}$ is distributed according to $\mathsf{Sh}(m_{(i,j)})$. Such a "patching" is possible by t-privacy of $(\mathsf{Sh}, \mathsf{Rec})$.
   c) For each honest $\mathcal{P}_k^\ell$ and corrupt R_j, $\mathcal{S}$ delivers $(s_{(1,j),k}, \dots, s_{(n,j),k})$ to $\mathcal{A}$ as message that $\mathcal{P}_k^\ell$ intends to send to R_j over f_{FM}^n.
   d) For each corrupt $\mathcal{P}_k^\ell$, $\mathcal{S}$ receives $(s_{(1,j),k}, \dots, s_{(n,j),k})$ from $\mathcal{A}$ as message $\mathcal{P}_k^\ell$ intends to send to each $\mathsf{R}_j, j \in [n]$ over f_{FM}^n. $\mathcal{S}$ updates the values of each $s_{(i,j),k}$ that was changed in this step.
   e) For each $\mathcal{P}_k^\ell \in \mathcal{L}_\ell$ and corrupt R_j, $\mathcal{S}$ delivers $(s_{(1,j),k}, \dots, s_{(n,j),k})$ as message from $\mathcal{P}_k^c$ to R_j over f_{FM}^n.
3. For each corrupt S_i, compute $m_{(i,j)} = \mathsf{Rec}(s_{(i,j),1}, \dots, s_{(i,j),n})$ as the message for R_j. Simulator sends $m_{(i,j)}$ to f_{FM}^n (from $\mathcal{L}_0$ to $\mathcal{L}_d$) as the message S_i intends to send to R_j. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

Figure 8.20: Simulator $\mathcal{S}$

We now argue that $\mathcal{S}$ satisfies Eq. (8.1). We separately analyze the outputs of honest parties and view of the adversary in both scenarios.

In the protocol, an honest R_j outputs $\mathsf{Rec}(\hat{s}_{(i,j),1}, \dots, \hat{s}_{(i,j),n})$ as the message from a sender S_i. In the simulation, output of R_j is the input of an honest S_i, but if S_i is corrupt, it is $\mathsf{Rec}(\hat{s}_{(i,j),1}, \dots, \hat{s}_{(i,j),n})$ (as chosen by $\mathcal{S}$). In both scenarios, $\hat{s}_{(i,j),k}$ coincides with $s_{(i,j),k}$ sent by S_i to $\mathcal{P}_k^\ell$ over f_{FM}^n from $\mathcal{L}_0$ to $\mathcal{L}_\ell$ if $\mathcal{P}_k^\ell$ is honest, and with $\hat{s}_{(i,j),k}$ sent by $\mathcal{P}_k^\ell$ to R_j over f_{FM}^n from $\mathcal{L}_\ell$ to $\mathcal{L}_d$ if $\mathcal{P}_k^\ell$ is corrupt. This directly implies that the output of R_j in both scenarios coincide when S_i is corrupt. When S_i is honest, $(s_{(i,j),1}, \dots, s_{(i,j),n})$ are shares of its input and $\hat{s}_{(i,j),k}$ coincides with $s_{(i,j),k}$ for at least $n - t$ distinct values of k (since there are at most t corrupt parties in $\mathcal{L}_\ell$). Hence, by the robustness of the secret sharing scheme, the output of R_j coincides with the input of an honest S_i.

The view of $\mathcal{A}$ consists of the messages sent by corrupt parties in $\mathcal{L}_0$, messages sent and

received by corrupt parties in $\mathcal{L}_\ell$ and the messages received by corrupt parties in $\mathcal{L}_d$. Note that, in step 3.(a). of the simulation, $(s_{(i,j),1}, \ldots, s_{(i,j),n})$ is sampled according to $\mathsf{Sh}(m_{(i,j)})$ for any honest S_i and corrupt R_j. Using this observation, and inspecting the protocol and its simulation, we conclude that the view accumulated by $\mathcal{A}$ in both scenarios differ *only* in the joint distribution of messages sent by each honest S_i to corrupt parties in $\mathcal{L}_\ell$ which they are expected to forward to honest receivers. Formally, the difference is in the generation of the joint distribution of $(s_{(i,j),k})$ for triples (i,j,k) such that $\mathsf{S}_i, \mathsf{R}_j$ are honest and $\mathcal{P}_k^\ell$ is honest. In the protocol, honest S_i samples $(s_{(i,j),1}, \ldots, s_{(i,j)}, n)$ according to $\mathsf{Sh}(m_{(i,j)})$ for (honest) R_j and sends $s_{(i,j),k}$ to the corrupt party $\mathcal{P}_k^\ell$. Whereas, in the simulation, for each honest S_i and honest R_j, $\mathcal{S}$ samples $(s_{(i,j),1}, \ldots, s_{(i,j)}, n)$ according to $\mathsf{Sh}(m)$ (for some arbitrary m) and sends $s_{(i,j),k}$ to the corrupt party $\mathcal{P}_k^\ell$. But, there are at most t corrupt parties in $\mathcal{L}_\ell$. Since the secret sharing scheme is t-robust, any set of t shares is identically distributed irrespective of the secret. Hence, we conclude that the distribution of the view of $\mathcal{A}$ (jointly, with outputs of honest parties) coincide in both scenarios, concluding the proof. □

8.10 Layered MPC based on CNF Secret Sharing

8.10.1 Proof of Lemma 8.4

Lemma 8.4. *For any $T \in \mathcal{T}$, the following properties hold for any weak future multicast protocol with U_T as intermediaries when executed in the presence of any adversary $\mathcal{A}$:*

(a) There exists $\hat{r}$ such that all honest receivers in R output $\hat{r}$ at the end of the protocol. Furthermore, if S is honest, $\hat{r} = r$.

(b) If S, and all intermediaries and receivers are honest, for any $r, r' \in M$,

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Proof. By Lemma 8.3:(1), every honest party in $\mathcal{L}_3$ computes the same output at the end of Π_{Add}. Hence, all ($n-t$ or more) honest parties broadcasts a complaint (or refrain from doing so) in unison. In the former case, all receivers get $\hat{r} = \mathsf{Rec}(\hat{r}_1, \ldots, \hat{r}_n)$ from each honest party in $\mathcal{L}_4$ and output the unique value sent by at least $n-t$ parties. Thus, all honest receivers output $\hat{r}$. When S is honest, by Corollary 8.1:(1), in every Π_{FM} initiated in step (i).2 of $\Pi_{\mathsf{weak\text{-}FMcast}}$ from S to $\mathcal{P}_i^3$, the output $\hat{r}_i$ coincides with sender's input r_i. Hence, each $\mathcal{P}_j^4, j \in [n]$ receives $\hat{r}_i = r_i$ from each honest $\mathcal{P}_i^3$ in the previous level. Consequently, $\hat{r} = \mathsf{Rec}(r_1, \ldots, r_n) = r$ by t-robustness of the secret sharing scheme. Thus, (a) holds in this case.

Suppose all honest parties in $\mathcal{L}_3$ refrain from registering complaints. Each receiver in R gets $(\hat{r}_i^j, j \in [n])$ from every honest $\mathcal{P}_i^4$ and computes the output according to step (v).2. By Corollary 8.1:(1), in each Π_{FM} (initiated in step (ii).2) from an honest sender $\mathcal{P}_j^1$ to an honest receiver $\mathcal{P}_i^4$, the output $\hat{r}_i^j$ coincides with the input r_i^j. Hence, by t-robustness of secret sharing, all honest receivers correctly recover $\mathsf{Rec}(\hat{r}_1^j, \ldots, \hat{r}_n^j) = r^j$ for each honest

$\mathcal{P}_j^1$. Furthermore, if every honest party in layer 3 refrains from registering a complaint, then the output of Π_{Add} computing $r^j - r^{j'}$ must have been zero for every pair $j, j' \in T$. By Lemma 8.3:(1), this specifically implies that r^j and $r^{j'}$ are identical for any pair of honest parties $\mathcal{P}_j^1$ and $\mathcal{P}_{j'}^1$. Thus, there exists r' such that $r^j = r'$ for all honest $\mathcal{P}_j^1$. Finally, when S is honest, r' coincides with r, the input of S. Since the set of $n-2t$ or more honest parties forms a majority of the intermediaries, all honest receivers output r'. This concludes the proof of (a).

Before proving (b), we informally argue that the sender's secret is not leaked when intermediaries and receivers are honest. Assume that the statements about security of Π_{FM} and Π_{Add} in Corollary 8.1 and Lemma 8.3 hold even when they are composed as in $\Pi_{\mathsf{weak\text{-}FMcast}}$. Since $r^j = r$ for every intermediary $\mathcal{P}_j^1$, $r^j - r^{j'} = 0$ for every pair j, j'. Consequently, every honest party in $\mathcal{L}_3$ refrain from broadcasting a complaint. Hence, the view of $\mathcal{A}$ interacting with $\Pi_{\mathsf{weak\text{-}FMcast}}$ with honest sender, intermediaries and receivers consists of the shares of r sent to the corrupt parties in $\mathcal{L}_3$ by S using Π_{FM} in step (i).2, and the shares of $r^j (= r)$ sent to corrupt parties in $\mathcal{L}_4$ by $\mathcal{P}_j^1$ using Π_{FM} initiated in step (ii).2. But, since the secret sharing scheme is t-secure, these shares do not reveal any information about r to the adversary. Thus, r remains private from the adversary.

We proceed to formally prove Lemma 8.4:(b) using a hybrid argument. Fix $r, r' \in M$ where $r \neq r'$. Let Θ (resp. Θ') be the protocol $\Pi_{\mathsf{weak\text{-}FMcast}}$ with $U_T = \{\mathcal{P}_i^1 : i \in T\}$ as intermediaries and r (resp. r') as input of S as described in Figure 8.8. We progressively transform Θ to obtain Θ' and argue that the adversary's view is identically distributed across each of these transformations, proving (b).

Transformation $\Theta \to \Theta_1$. Suppose w.l.o.g. suppose $\mathcal{P}_i^3$ is honest if $i \in [k]$, where $k \geq n - t$. Consider a sequence of protocols $(\Theta_{0,i})_{0 \leq i \leq k}$, where $\Theta_{0,0}$ is identical to Θ. For $i \geq 1$, $\Theta_{0,i}$ is obtained by replacing $(r_1, \ldots, r_n)$ in step (i).2 with $(r'_1, \ldots, r'_i, r_{i+1}, \ldots, r_n)$, where $(r'_1, \ldots, r'_k, r_{k+1}, \ldots, r_n)$ is distributed according to $\mathsf{Sh}(r')$. Note that $(r'_1, \ldots, r'_k)$ can be sampled in this manner since the secret sharing scheme is t-secure and $k \geq n - t$. Finally, define $\Theta_1 = \Theta_{0,k}$. Notice that, in Θ_1, S sends a secret sharing of r' in step (i).2. The following claim suffices to prove that

$$\mathsf{ADVR}_{\Theta,\mathcal{A}}(\perp) \equiv \mathsf{ADVR}_{\Theta_1,\mathcal{A}}(\perp). \tag{8.2}$$

Here the argument is $\perp$ because neither protocols take any inputs.

Claim 8.1. *For any $1 \leq \ell \leq k$,*

$$\mathsf{ADVR}_{\Theta_{0,\ell-1},\mathcal{A}}(\perp) \equiv \mathsf{ADVR}_{\Theta_{0,\ell},\mathcal{A}}(\perp).$$

The only difference between $\Theta_{0,\ell-1}$ and $\Theta_{0,\ell}$ is that the sender's input in the instance ℓ of Π_{FM} from S to $\mathcal{P}_\ell^3$ is r_ℓ in the former and r'_ℓ in the latter. Since the sender and all intermediaries are honest, as we previously observed, all honest parties in $\mathcal{L}_3$ refrain from broadcasting a complaint and, more importantly, refrain from broadcasting the output of Π_{FM} from S initiated in step (i).2. We will show that, if Eq. (8.2) does not hold, then Corollary 8.1:(2) is not satisfied for Π_{FM} from S to $\mathcal{P}_\ell^3$; a contradiction.

Adversary $\mathcal{A}'$ with auxiliary input $\{r_i, i \in [n]\}, \{r'_i, i \in [k]\}$ non-adaptively corrupting the same set of parties that $\mathcal{A}$ corrupted in $\mathcal{L}_1$ and $\mathcal{L}_2$ interacts with an execution of Π_{FM} from honest S to honest $\mathcal{P}^3_i$ as follows:

- Let $\mathcal{I}_i \subset \mathcal{L}_i$, $0 \leq i \leq 5$ be the set of parties corrupted by $\mathcal{A}$. $\mathcal{A}'$ generates dummy corrupt parties $\mathcal{I}_i = \{\mathcal{P}^i_j : \mathcal{P}^i_j \in \mathcal{I}_i, i \in \{0,3,4,5\}\}$ and initialize their random tapes.

- $\mathcal{A}'$ hands over the control of corrupt parties $\mathcal{I}_i$, $i = 1, 2$ and dummy corrupt parties $\mathcal{I}_i$, $i \in \{0,3,4,5\}$ to $\mathcal{A}$ which it internally invokes. $\mathcal{A}'$ also generates dummy honest parties $\mathcal{H}_i = \mathcal{L}_i \setminus \mathcal{I}_i$ for each $0 \leq i \leq 5$.

- Using the values of r_i, r'_i, $\mathcal{A}'$ emulates dummy honest parties $\mathcal{H}_i$ for each $0 \leq i \leq 5$ executing $\Theta_{0,\ell-1}$ and interacts with $\mathcal{A}$. However, it does not emulate the instance of Π_{FM} from S to $\mathcal{P}^3_\ell$ initiated in step (i).2. of $\Theta_{0,\ell-1}$. Instead, $\mathcal{A}'$ interacts with the real execution of Π_{FM} in parallel, and forwards the messages from (and to) the honest parties in the real execution to (and from) the corrupt parties $\mathcal{I}_1 \cup \mathcal{I}_2$ who $\mathcal{A}'$ has handed over to $\mathcal{A}$. We stress that the output of real $\mathcal{P}^3_\ell$ at the end of the execution Π_{FM} from S to $\mathcal{P}^3_\ell$ is not needed by $\mathcal{A}'$ for this emulation. This is because, in $\Theta_{0,\ell-1}$ all honest parties, specifically $\mathcal{P}^3_\ell$, refrains from revealing the output of Π_{FM} from S (from step (i).2).

- At the end of $\Theta_{0,\ell-1}$, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ outputs.

In the instance of Π_{FM} from S to $\mathcal{P}^3_\ell$ in step (i).2, if the input S is r_ℓ, the above interaction is identical to the interaction with parties executing $\Theta_{0,\ell-1}$ for adversary $\mathcal{A}$ corrupting $\mathcal{I}_i, 0 \leq i \leq 5$. This can be verified as follows: In the former, all instructions in $\Theta_{0,\ell-1}$ except the ones that are part of Π_{FM} from S to $\mathcal{P}^3_\ell$ are carried out by dummy honest parties and corrupt parties controlled by $\mathcal{A}$. Whereas, in the latter, they carried out by the real honest parties and corrupt parties controlled by $\mathcal{A}$. But, $\mathcal{A}'$ emulates all the dummy parties honestly using freshly chosen randomness in the former case just as the honest parties will in the latter case. Thus,

$$\mathsf{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}'}(r_\ell) \equiv \mathsf{ADVR}_{\Theta_{0,\ell-1},\mathcal{A}}(\perp).$$

As previously observed, $\Theta_{0,\ell-1}$ and $\Theta_{0,\ell}$ differ only in the input of the sender in Π_{FM} from S to $\mathcal{P}^3_\ell$. Hence, in the instance of Π_{FM} from S to $\mathcal{P}^3_\ell$ in step (i).2, if the input S is r'_ℓ, the above interaction is identical to the interaction with parties executing $\Theta_{0,\ell}$ for adversary $\mathcal{A}$ corrupting $\mathcal{I}_i, 0 \leq i \leq 5$. Thus,

$$\mathsf{ADVR}_{\Pi_{\mathsf{FM}},\mathcal{A}'}(r'_\ell) \equiv \mathsf{ADVR}_{\Theta_{0,\ell},\mathcal{A}}(\perp).$$

But, since S and $\mathcal{P}^3_\ell$ are honest, by Corollary 8.1,

$$\mathsf{ADVR}_{\Pi_{\mathsf{S}},\mathcal{A}'}(m'_j) \equiv \mathsf{ADVR}_{\Pi_j,\mathcal{A}}(\perp).$$

This concludes the proof of the claim. □

Transformation $\Theta_1 \to \Theta_2$. We construct Θ_2 from Θ_1 by setting $r^j = r'$ (instead of $r^j = r$) for each party $\mathcal{P}^1_j$ in step (ii).1. Intuitively, since all intermediaries are honest, parties in $\mathcal{L}_3$ only learn $r^j - r^{j'}$, and hence the view of $\mathcal{A}$ will be identical in both cases. To formally

prove this, we again consider a sequence of hybrids which progressively transforms Θ_1 to Θ_2. In every new hybrid, we change the inputs used by a new pair of honest parties $\mathcal{P}_j^1, \mathcal{P}_{j'}^1$ $(j < j')$ to Π_{Add} for computing $r^j - r^{j'}$ (in step (i).1) from $r^j = r, -r^j = -r$, respectively, to $r^j = r', -r^j = -r'$. Clearly, the final protocol in this sequence is exactly Θ_2. The difference between any pair of consecutive hybrids is the replacement of the inputs of a specific pair of honest parties $\mathcal{P}_j^1, \mathcal{P}_{j'}^1$ $(j < j')$ to Π_{Add} from $r^j = r, -r^j = -r$, respectively, to $r^j = r', -r^j = -r'$. We argue that the view of $\mathcal{A}$ remains identically distributed across any pair of consecutive hybrids by appealing to Lemma 8.3 using the same line of arguments as in the above claim. We thus obtain,

$$\mathsf{ADVR}_{\Theta_1,\mathcal{A}}(\perp) \equiv \mathsf{ADVR}_{\Theta_2,\mathcal{A}}(\perp). \tag{8.3}$$

Transformation $\Theta_2 \to \Theta_3$. To obtain Θ_3 from Θ_2, for each $i \in T$, we replace $(r_1^i, \ldots, r_n^i)$ in step (ii).2 with $(s_1^i, \ldots, s_n^i)$, where $(s_1^i, \ldots, s_n^i)$ is distributed according to $\mathsf{Sh}(r')$ and $r_j^i = s_j^i$ for each j such that $\mathcal{P}_j^4 \in \mathcal{L}_4$ is corrupt. This is possible since the secret sharing scheme is t-secure and there are at most t corrupt parties in $\mathcal{L}_4$. Arguing that the adversary's view remains the same across this transformation is similar to arguing the same for transformation from Θ_0 to Θ_1. We thus obtain,

$$\mathsf{ADVR}_{\Theta_2,\mathcal{A}}(\perp) \equiv \mathsf{ADVR}_{\Theta_3,\mathcal{A}}(\perp). \tag{8.4}$$

Θ_3 can be transformed to Θ' by replacing r with r' in step (i).1. Note that this does not make any difference to the step (ii), which has already been modified to replace r with r'. Thus, from Eqs. (8.2) to (8.4), we conclude that

$$\mathsf{ADVR}_{\Theta,\mathcal{A}}(\perp) \equiv \mathsf{ADVR}_{\Theta',\mathcal{A}}(\perp).$$

This proves (b) concluding the proof. □

8.10.2 Constructing $\mathcal{S}$ for Theorem 8.2

Theorem 8.2. *There is a secure $(n, t, 5)$-layered protocol realizing future multicast with input client S and output clients in R.*

Proof. When S is corrupt, $\mathcal{S}$ emulates the honest parties according to the instructions in Π_{FMcast} and interact with the adversary. Observe that none of the honest parties have inputs since S is the only party with input. For each $T \in \mathcal{T}$, $\mathcal{S}$ extracts an input r_T of S in $\Pi_{\mathsf{weak\text{-}FMcast}}$ with intermediaries U_T as described below: We observed in the proof of statement (a) of Lemma 8.4 that, at the end of $\Pi_{\mathsf{weak\text{-}FMcast}}$, all receivers output $\hat{r}$ computed in step (iv).1 of the protocol if all the honest parties in $\mathcal{L}_3$ broadcasted a complaint, and otherwise–i.e., if no honest party in $\mathcal{L}_3$ broadcasted a complaint–all receivers output the unique value that was received by all honest intermediaries. Hence, if all emulated honest parties in $\mathcal{L}_3$ broadcasted a complaint, $\mathcal{S}$ chooses r_T to be $\hat{r}$ (computed by emulated honest parties), and otherwise it choose r_T as the unique value received by all emulated honest intermediaries. Finally, it sends $\sum_{T \in \mathcal{T}} r_T$ to f_{FMcast} as the input of S. Security follows from statement (a) in Lemma 8.4.

Next, suppose S is honest. In this case, $\mathcal{S}$ emulates all the honest parties and interacts with $\mathcal{A}$. For this, $\mathcal{S}$ sets the input of S to an arbitrary fixed message $m^* \in M$. Since the output of all honest receivers coincides with the input of the sender, it suffices to show that the output of $\mathcal{A}$ is identically distributed while interacting with Π_{FMcast} or with $\mathcal{S}$.

Let $T^* \in \mathcal{T}$ such that $U_{T^*} = \{\mathcal{P}_i^1 : i \in T^*\}$ are honest. The shares $(r_T, T \in \mathcal{T} \setminus \{T^*\})$ used in the simulation is identically distributed as $(r_T, T \in \mathcal{T} \setminus \{T^*\})$ used in Π_{FMcast}. This is because, in step 1 of Π_{FMcast}, $(r_T, T \in \mathcal{T})$ is chosen to be an additive secret sharing of the input of S. Hence, for each $T \in \mathcal{T} \setminus \{T^*\}$, execution of $\Pi_{\text{weak-FMcast}}$ with U_T as intermediaries in the presence of $\mathcal{A}$ is identical in both the simulation and in Π_{FMcast}. However, in $\Pi_{\text{weak-FMcast}}$ with U_{T^*} as intermediaries, the input of S is $m^* - \sum_{T \in \mathcal{T} \setminus \{T^*\}} r_T$ in the former and $m - \sum_{T \in \mathcal{T} \setminus \{T^*\}} r_T$ in the latter.

If all the receivers are honest, by statement (b) of Lemma 8.4, the view of the adversary in $\Pi_{\text{weak-FMcast}}$ with U_{T^*} as intermediaries is identical in both the simulation and in Π_{FMcast}, proving security.

However, when there are corrupt parties in R, the view of $\mathcal{A}$ additionally contains the view of the corrupt receivers. The view of corrupt receivers in $\Pi_{\text{weak-FMcast}}$ with U_{T^*} as intermediaries needs to be made consistent with $r_{T^*} = m - \sum_{T \in \mathcal{T} \setminus \{T^*\}} r_T$ even though $\mathcal{S}$ chose $r_{T^*} = m^* - \sum_{T \in \mathcal{T} \setminus \{T^*\}} r_T$ in the simulation. To ensure this, $\mathcal{S}$ tweaks the messages from the emulated honest parties in $\mathcal{L}_4$ to the corrupt receivers in R as follows: $\mathcal{S}$ receives the input m of S from f_{FMcast}. In $\Pi_{\text{weak-FMcast}}$, for each intermediary $\mathcal{P}_i^1 \in U_{T^*}$, $\mathcal{S}$ samples $(\hat{r}_1^i, \ldots, \hat{r}_n^i)$ according to distribution $\mathsf{Sh}(m - \sum_{T \in \mathcal{T} \setminus \{T^*\}} r_T)$ conditioned on $\hat{r}_j^i = r_j^i$ for each j such that $\mathcal{P}_j^4$ in $\mathcal{L}_4$ is corrupt. This is possible since the secret sharing scheme is t-secure and there are at most t corrupt parties in each layer. In step (iv).2 (note that parties choose this step since all intermediaries and S are honest as observed in the proof of Lemma 8.4), each emulated honest party sends $\hat{r}_j^i$ instead of r_j^i. This ensures that the view of corrupt receivers in $\Pi_{\text{weak-FMcast}}$ with U_{T^*} as intermediaries is consistent with the input of S. This concludes the proof. □

8.10.3 Constructing $\mathcal{S}$ for Theorem 8.3

Theorem 8.3. *The protocol in Figure 8.10 executed in parallel realizes $f_{\text{parallel-VSS}}$ with perfect t-security for $t < n/3$ by consuming 5 layers, and by communicating $\binom{n}{t}^3 \cdot O(n^2)$ field elements over the point-to-point channels and over the broadcast channels for each secret.*

Proof. We now formally describe the simulator. The parallel VSS protocol with $\mathcal{P}_i^0$ as dealer consists of $|\mathcal{T}|$ parallel multicast protocols. For $T \in \mathcal{T}$, dealer multicasts r_T^i to receivers $\mathcal{P}_j^5, j \in T$. The simulator emulates all the honest parties in each of these multicast protocols and interacts with the adversary. For this, each honest dealer (the honest dealers are the only honest parties with input) are initialized with uniformly random inputs for every multicast in which they act as sender. For each corrupt dealer $\mathcal{P}_i^0$, the simulator extracts $\hat{r}_T^i$ as the input in the multicast from $\mathcal{P}_i^0$ to receivers $\mathcal{P}_j^5, j \in T$ for each $T \in \mathcal{T}$. The simulator then sets $(\hat{r}_T^i)_{T \in \mathcal{T}}$ as the input of corrupt $\mathcal{P}_i^0$ to $f_{\text{parallel-VSS}}$.

We now argue that the view of the adversary is identically distributed when interacting with the simulator and with honest parties executing the protocol for parallel VSS. Suppose $\mathcal{P}_j^5, j \in T^*$ are honest (such T^* exists since the adversary corrupts at most t parties in each layer) and let $\mathcal{T}^* = \mathcal{T} \backslash \{T^*\}$. For any honest dealer $\mathcal{P}_i^0$ and any input to the dealer, the additive share $r^i_{T^*}$ of the input is sent only to honest receivers. Since the additive shares $(r^i_T)_{T \in \mathcal{T}^*}$ are uniformly distributed, the only difference between the real protocol and the simulation is that honest $\mathcal{P}_i^0$ with input m uses $r^i_{T^*} = m - \sum_{T \in \mathcal{T}^*} r^i_T$ in the real execution, whereas it uses $r^i_{T^*} = m^* - \sum_{T \in \mathcal{T}^*} r^i_T$ in the simulation, where m^* is the arbitrary input that the simulator assigns to $\mathcal{P}_i^0$. Thus, security amounts to showing that the view of the adversary is identically distributed in the multicast protocol with honest sender and receivers irrespective of sender's input, which follows from Theorem 8.2:(b). Note that, this can be shown to hold for several parallel executions of multicasts using a hybrid argument in which the inputs in the multicast protocols are progressively changed one at a time to move from one set of inputs to a different set of inputs to the senders in the parallel multicasts. Finally, it needs to be shown that the extracted inputs for the corrupt parties are independent of the shares chosen by the honest parties in VSS. This follows from the fact that the view of corrupt parties in $\mathcal{L}_0, \ldots, \mathcal{L}_4$ does not reveal the message that an honest sender multicasts to corrupt receivers and shown in the proof of Theorem 8.2. □

8.10.4 Details omitted from Section 8.4.3

A t-secure protocol for multiplication *in the standard setting*

INPUTS: Party $P_i, i \in [n]$ has inputs (r_i, s_i), where $(r_i)_{i \in [n]}$ and $(s_i)_{i \in [n]}$ are CNF secret shares of r and s, respectively.
DEFINITIONS: Let $\mathcal{T} = \{T_1, \ldots, T_N\} = \{T \subset [n] : |T| = n - t\}$.
For each $i \in [n]$, $s_i = (\gamma_j : i \in T_j, j \in [N])$
and $r_i = (\lambda_j : i \in T_j, j \in [N])$, where $\sum_{j \in [N]} \gamma_j = r$ and $\sum_{j \in [N]} \lambda_j = s$.

1. Party $P_i, i \in [n]$ verifiably shares $\gamma_j \cdot \lambda_{j'}$ for each (j, j') such that P_i has both γ_j and $\lambda_{j'}$. Party P_i has γ_j (resp. λ_j) if $i \in T_j$.
2. For each $(j, j') \in [N] \times [N]$, let $P_i, i \in I$ be the set of parties that secret shared $\gamma_j \cdot \lambda_{j'}$. Fix $i^* \in I$; for all $i \in I, i \neq i^*$, publicly check if VSS of $\gamma_j \cdot \lambda_{j'}$ provided P_{i^*} and P_i are equal. This amounts to securely computing the differences between the values CNF shared by both parties. If true, take the VSS provided by P_{i^*} as the secret sharing of $\gamma_j \cdot \lambda_{j'}$. Otherwise, disclose γ_j and $\lambda_{j'}$ by having all parties reveal γ_j and $\lambda_{j'}$. The secret sharing of $\gamma_j \cdot \lambda_{j'}$ is trivially computed from $\gamma_j \cdot \lambda_{j'}$.
3. Each party computes its share of $r \cdot s$ as the sum of their respective shares of $\gamma_j \cdot \lambda_{j'}$ for all $j, j' \in [N] \times [N]$.

Figure 8.21: A t-secure protocol for multiplication

8.10.5 Proof of Lemma 8.5

Lemma 8.5. *For any $j, j' \in [N]$, the following properties hold for $\Pi_{j,j'}$ when executed in the presence of an adversary $\mathcal{A}$:*

(a) *There exists* $(\delta_k)_{k\in[N]}$ *such that* $\sum_{k=1}^{N}\delta_k = \lambda_j\gamma_{j'}$*, and each honest party* $\mathcal{P}_i^7, i \in [N]$ *outputs* $(\delta_k)_{k:i\in T_k}$ *at the end of* $\Pi_{j,j'}$.

(b) *Suppose parties* $\mathcal{P}_i^0, i \in H$ *are honest, then for any* a, b, a', b',

$$\mathsf{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a, \lambda_{j'} = b) \equiv \mathsf{ADVR}_{\Pi_{j,j'},\mathcal{A}}(\gamma_j = a', \lambda_{j'} = b').$$

Proof. By the linearity of CNF secret sharing, and its t-robustness, all honest parties in $\mathcal{L}_6$ obtain the same value for $\sum_{l\in[N]}\delta_l - (\gamma\lambda)^i$ for each $i \neq i^*$. Hence, all ($n-t$ or more) honest parties broadcasts a complaint (or refrain from doing so) in unison. In the former case, all honest parties in $\mathcal{L}_7$ recover $\gamma = \gamma_j$ and $\lambda = \lambda_{j'}$. This can be seen as follows: all parties in $\mathcal{L}_7$ receive $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$ from each party in $\mathcal{P}_k^6, k \in [n]$. Since there are at most t corrupt parties in $\mathcal{L}_6$, by t robustness of $(\mathsf{Sh}, \mathsf{Rec})$, γ^i computed as $\mathsf{Rec}(\gamma^{i,1}, \ldots, \gamma^{i,n})$ coincides with γ for each i such that $\mathcal{P}_i^0$ is honest. Since there are at least $n - 2t(> t)$ honest parties among $\mathcal{P}_i^0, i \in T_j$, all honest parties correctly recover γ. Similarly, all honest parties recover λ. But then, all honest parties choose $\delta_k = \gamma\lambda$ if $k = 1$ and $\delta_k = 0$ otherwise. Thus, (a) holds in this case.

When all honest parties in $\mathcal{L}_6$ refrain from registering complaints, we will show that $\sum_{l=1}^{N}\delta_l = \gamma\lambda$. Furthermore, each party $\mathcal{P}_i^7$ in the output layer correctly recover δ_k for each k such that $i \in T_k$. These two observations directly imply (a). We justify the first observation as follows: since $|I| = |T_j \cap T_{j'}| \geq t+1$, there exists $\hat{i} \in I$ such that party $\mathcal{P}_{\hat{i}}^0$ is honest, and hence correctly carries out VSS. Thus, depending on the value of $\hat{i}$, either $\sum_{l=1}^{N}\delta_l = \gamma\lambda$ (when $\hat{i} = i^*$) or $(\lambda\gamma)^{\hat{i}} = \gamma\lambda$ (when $\hat{i} \neq i^*$). In either case, we get $\sum_{l=1}^{N}\delta_l = \gamma\lambda$, since all honest parties in $\mathcal{L}_6$ refrain from registering complaints only if,

$$\sum_{l=1}^{N}\delta_l - (\lambda\gamma)^i = 0, \forall i \in I, i \neq i^*.$$

For each k such that $i \in T_k$, party $\mathcal{P}_i^7, i \in [n]$ correctly receives a CNF share of δ_k from each honest $\mathcal{P}_j^5$ via Future Messaging (see step (ii).2). $\mathcal{P}_i^7$ can correctly reconstruct δ_k from these shares; this follows from CNF VSS being t-robust and the number of corruption is $\mathcal{L}_5$ being at most t. This concludes the proof of (a).

Before proving (b), we informally argue that $\Pi_{j,j'}$ does not leak γ and λ when $\mathcal{P}_i^0$ is honest for each $i \in I$. Observe that, when parties $\mathcal{P}_i^0$ for all $i \in I$ are honest, irrespective of the value of γ and λ,

$$\sum_{l=1}^{N}\delta_l - (\lambda\gamma)^i = 0, \forall i \in I, i \neq i^*.$$

Hence, each honest party $\mathcal{P}_k^6$ refrains from broadcasting a complaint, or broadcasting $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$. Assume that the statements about security of Π_{FM} and Π_{VSS} hold even when they are composed as in $\Pi_{j,j'}$. Then, the view of $\mathcal{A}$ interacting with $\Pi_{j,j'}$ consists of

1. The shares of λ and γ sent to the corrupt parties in $\mathcal{L}_6$ by parties $\mathcal{P}_i^0, i \in T_j$ and $\mathcal{P}_i^0, i \in T_{j'}$ using Future Messaging in step (i).2 and (i).3, respectively.

2. Let $\mathcal{P}_i^7, i \in C_7$ be the corrupt parties in $\mathcal{L}_7$. Each corrupt party $\mathcal{P}_i^7$ receives CNF shares of δ_k for each k such that $i \in T_k$ from parties in $\mathcal{L}_5$ via Future Messaging initiated in step (ii).2. From this, $\mathcal{A}$ can recover δ_k for each $k \in [N] \setminus \{k^*\}$, where k^* such that $T_{k^*} \subseteq [n] \setminus C_7$. There exists such a $k^* \in [N]$ since $|C_7| \leq t$ and $\{T_1, \ldots, T_N\}$ consists of all subsets of $[n]$ of size $n - t$.

3. View of corrupt parties in $\mathcal{L}_6$ during the secure computation of $\sum_{l=1}^{N} \delta_l - (\lambda\gamma)^i$ for each $i \in I, i \neq i^*$.

Here (1) does not reveal γ and λ since (Sh, Rec) is a t-secure secret sharing scheme and there are at most t corrupt parties in $\mathcal{L}_6$. Since $\delta_1, \ldots, \delta_N$ are sampled to be additive secret shares of $\gamma\lambda$, $(\delta)_{k\in[N]\setminus\{k^*\}}$ revealed to adversary in (2) are uniformly random irrespective of the value of $\gamma\lambda$. Finally, since each $\mathcal{P}_i^0, i \in I, i \neq I^*$ provides a fresh VSS of $\gamma\lambda$ to $\mathcal{L}_5$ and $\sum_{l=1}^{N} \delta_i = \gamma\lambda$, irrespective of the value of $\gamma\lambda$ and $(\delta_k)_{k\in[N]\setminus\{k^*\}}$, the view of parties in $\mathcal{L}_6$ during secure computation of $\sum_{l=1}^{N} \delta_i - (\lambda\gamma)^i$ is distributed like the CNF VSS of 0, independently, for each $i \in I \setminus \{i^*\}$. Thus, the view of the adversary is identically distributed irrespective of the value of γ and λ, proving (b).

We proceed to formally proving (b) using a hybrid argument. Fix a, b, a', b'. Let $\mathcal{P}_i^7, i \in C_7$ be the corrupt parties in $\mathcal{L}_7$ and let $k^* \in [N]$ such that $T_{k^*} \cap C_7 = \emptyset$. Let Θ be the protocol obtained from $\Pi_{j,j'}$ when (γ, λ) is set to (a, b) in every step of the protocol. Let Θ' be the protocol obtained from $\Pi_{j,j'}$ when (γ, λ) is set to (a', b') in every step of the protocol. We progressively transform Θ to obtain Θ' and argue that the adversary's view is identically distributed across each of these transformations, proving (b).

Transformation $\Theta \to \Theta_1$. Θ_1 is obtained from Θ by setting γ to a and λ to b in steps (i).2 and (i).3, respectively. As we already observed, since $\mathcal{P}_i^0$ is honest for each $i \in I$, each honest party $\mathcal{P}_k^{a+6}$ refrains from broadcasting a complaint, or broadcasting $(\gamma^{i,k})_{i\in T_j}$ and $(\lambda^{i,k})_{i\in T_{j'}}$. Hence, the view of the adversary in Θ and Θ_1 are identical; this uses the same line of argument used for the transformation from Θ to Θ_1 in the proof of Lemma 8.4.

Transformation $\Theta_1 \to \Theta'$. Θ' is obtained from Θ_1 by making the following replacements in step (i).1: $\mathcal{P}_{i^*}^0$ verifiably secret shares $\delta_{k^*} - (ab) + (a'b')$ instead of δ_{k^*} (δ_k for $k \neq k^*$ remain unchanged); for each $i \in I, i \neq i^*$, $\mathcal{P}_i^0$ verifiably secret shares $(a'b')$ instead of (ab). Since $(\delta_k)_{k\in[N]\setminus\{k^*\}}$ along with $\delta_{k^*} - (ab) + (a'b')$ forms an additive secret sharing of $(a'b')$, Θ' is $\Pi_{j,j'}$ with (γ, λ) set to (a', b'). By inspecting the protocol, appealing to security of Π_{VSS} and Π_{FM}, it is easy to see that the view of $\mathcal{A}$ is identically distributed. This proves (b) concluding the proof. □ □

8.11 Layered MPC based on Shamir Secret Sharing

8.11.1 Details omitted from Section 8.5.1

8.11.1.1 Proof of Lemma 8.6

Lemma 8.6. *The following properties hold for an execution of* Π *in the presence of a layered adversary* $\mathcal{A}$*:*

(a) Let $G \subseteq [n]$ *such that* $\mathcal{P}_i^1$ *is honest if and only if* $i \in H_1$*. There exist polynomials* $\hat{g}(x)$ *and* $\hat{g}_i(x), i \in H_1$*, each of degree at most* t*, such that* $\hat{g}_i(0) = \hat{g}(i)$ *and* α_i^k *output by each honest party* $\mathcal{P}_k^5$ *coincides with* $\hat{g}_i(k)$*. Furthermore, if* S *is honest,* $\hat{g}(x) = F(x, 0)$*.*

(b) If S *is honest, for any* $r, r' \in \mathbb{F}$*,*

$$\mathrm{ADVR}_{\Pi,\mathcal{A}}(r) \equiv \mathrm{ADVR}_{\Pi,\mathcal{A}}(r').$$

Proof. Suppose S is honest. Then, all the polynomials received by parties in $\mathcal{L}_1$ are consistent; i.e, $f_i(j) = g_j(i)$ for all $i, j \in [n]$. For any $\{i, j\} \in S$, by the correctness of future messaging, $F(i, j) = g_i(j)$ and $F(j, i) = f_i(j)$, whenever $\mathcal{P}_i^1$ is honest. Thus, if $i \in B$ (defined in (iv).3), then $\mathcal{P}_i^1$ is necessarily corrupt. Hence, revealing $f_i(x)$ and $g_i(x)$ for $i \in B$, provide no information to the adversary in addition to what it learned in $\mathcal{L}_1$. By correctness of future messaging, for each $i \in B$, $\hat{g}_i(x)$ and $\hat{f}_i(x)$ recovered in step (v).1 coincide with $f_i(x)$ and $g_i(x)$, respectively. Hence, for any honest $\mathcal{P}_j^1$, by correctness of future messaging, $\hat{f}_i(j) = g_j(i)$ and $\hat{g}_i(j) = f_j(i)$. In conclusion, $i \in B' \cup B$ only if $\mathcal{P}_i^1$ is corrupt. Thus, $|B \cup B'| \leq t$ and, hence, the dealer is not disqualified. Finally, by correctness of future messaging, for each $i \in H_1$, g_i^k is a valid Shamir secret share of $g_i(0)$ whenever $\mathcal{P}_k^5$ is honest, and $g_i(0) = F(i, 0)$ for all $i \in H_1$. This proves (a) when S is honest.

Suppose S is corrupt. In step (v).3, if $|B \cup B'| > t$, then (a) holds with $q(x)$ and $g_i(x)$, $i \in H_1$ as zero polynomials. When $|B \cup B'| \leq t$, there exist at least $t + 1$ distinct $i \in H_1$ such that $i \notin B \cup B'$.

Define $H_1 \setminus (B \cup B') = H_1'$. Let $i, j \in H_1'$. If $f_i(j) \neq g_j(i)$ or $g_i(j) \neq f_j(i)$, by correctness of Π_{Add}, $\{i, j\} \in S$. Furthermore, the purported $F(i, j)$ recovered in step (iv).1 does not coincide with $f_i(j)$ or with $g_j(i)$ if $f_i(j) \neq g_j(i)$. A similar condition holds when $g_i(j) \neq f_j(i)$. Hence, in this case, $\{i, j\} \cap B \neq \emptyset$; a contradiction. Hence, for every $i, j \in H_1'$, the polynomials received by $\mathcal{P}_i^1$ and $\mathcal{P}_j^1$ are pairwise consistent. Additionally, since $H_1' \cap B' = \emptyset$, for every $i \in B$, $\hat{f}_i$ and $\hat{g}_i$ recovered in step (v).1 satisfy $\hat{f}_i(j) = g_j(i)$ and $\hat{g}_i(j) = f_j(i)$ for all $j \in H_1'$. Given these observations, the following claim implies that there exists a unique bivariate polynomial $\hat{F}(x, y)$ with degree at most t in both variables such that, for every $i \in H_1'$, $f_i(x) = \hat{F}(x, i)$ and $g_i(y) = \hat{F}(i, y)$. This claim is proved in [AL17, Claim 5.3].

Claim 8.2. *Let* $f_i(x), g_i(y)$ *be polynomials of degree at most* t *for each* $i \in [m]$*, where* $m > t$*. Let* $\alpha_k, k \in [m]$ *be distinct non-zero elements in* $\mathbb{F}$*. If for all* $i, j \in [m]$*, it holds*

that $f_i(\alpha_j) = g_j(\alpha_i)$, then there exists a unique bivariate polynomial $F(x, y)$ with degree at most t in both variables such that $f_i(x) = F(x, \alpha_i)$ and $g_i(x) = F(\alpha_i, y)$ for all $i \in [m]$.

Finally, we need to argue that if $i \in H_1 \cap (B \cup B')$, then also $f_i(x)$ and $g_i(x)$ are consistent with $\hat{F}(x, y)$. Suppose $i \in H_1 \cap (B \cup B')$ such that $f_i(x) \neq \hat{F}(x, i)$. Since $f_i(x)$ is of degree at most t, there are at most t distinct $j \in H_1'$ such that $f_i(j) = \hat{F}(j, i) = g_j(i)$. In other words, there exists $j \in H_1'$ such that $f_i(j) \neq g_j(i)$. Then $\{i, j\} \in S$; but since $j \notin B$, necessarily, $i \in B$. But then, it must be the case that $\hat{f}_i$ and $\hat{g}_i$ broadcasted in step (v).1 are compatible with $f_j(x), g_j(x)$ for each $j \in H_1'$, otherwise, we get a contradiction. At this point, we conclude that $\hat{g}_i(0) = F(i, 0)$ for all $i \in H_1$. Furthermore, by correctness of future messaging, for each $i \in H_1$, $g_i^k(0)$ is a Shamir share of $\hat{g}_i(0)$ whenever $\mathcal{P}_k^5$ is honest. This proves (a).

We sketch the intuition behind (b). As previously observed, $f_i(x)$ and $g_i(x)$ are never revealed if $\mathcal{P}_i^1$ is honest. Hence, given the security of Π_{Add} and Π_{FM}, the adversary only learns the values of $F(x, y)$ on the polynomials that were revealed to corrupt parties in $\mathcal{L}_1$. A formal proof of (b) can be obtained by following a similar line of argument as in proving Lemma 8.4 (b). We leave this to the reader. □

8.11.2 Random secret sharing, resharing and reinforced secret sharing

Notation. In the sequel, we will use the following notations to denote secret sharing and their manipulations.

1. Let $s \in \mathbb{F}$ and let $f(x)$ be a polynomial of degree at most t such that $f(0) = s$. Suppose s has been secret shared on layer a using $f(x)$; i.e., each $P_i^a, i \in [n]$ gets $s_i = f(i)$. We denote this "state" by $\langle s, f(x)\rangle_a$.

2. We will denote the local addition of shares $\langle a, f(x)\rangle_a$ and $\langle b, g(x)\rangle_a$ by parties in $\mathcal{L}_a$ by $\langle a, f(x)\rangle_a + \langle b, g(x)\rangle_a$. By linearity of Shamir secret sharing, that $\langle a, f(x)\rangle_a + \langle b, g(x)\rangle_a = \langle a + b, g(x) + f(x)\rangle_a$.

3. Similarly, the local multiplication of shares $\langle a, f(x)\rangle_a$ by a constant $\alpha \in \mathbb{F}$ is denoted by $\alpha\langle a, f(x)\rangle_a$. Once again, by linearity, $\alpha\langle a, f(x)\rangle_a = \langle \alpha a, \alpha f(x)\rangle_a$.

4. $\mathcal{L}_a$, holding shares $\langle s, f(x)\rangle_a$, can privately reveal the secret s to a designated $\mathcal{P}_i^b$ for $b > a$ by securely communicating all the shares to the party. If $b > a+1$, this can be realized using future messaging. $\mathcal{P}_i^b$ can correctly recover s since the secret sharing is t-robust and future messaging with honest sender can be correctly recovered. We denote this process by $(a)_{\mathcal{P}_i^b} \Leftarrow \langle a, f(x)\rangle$.

5. $\mathcal{L}_a$, holding $\langle s, f(x)\rangle_a$, can reveal s to all parties in layer $b > a$ by communicating all the shares to each $\mathcal{P}_i^b, i \in [n]$. If $b > a + 1$, this can again be realized using future messaging to ensure that parties in layer c, for $a < c < b$, do not learn s. For the same reason as above, all parties in $\mathcal{L}_b$ will correctly recover s. We denote this process by $(s)_b \Leftarrow \langle s, f(x)\rangle_a$.

In this section, we use verifiable secret sharing to implement several useful primitives that we will use extensively in the secure implementation of secure multiplication and MPC. We will develop shorthand notations of the kind defined above for each primitive we define in this section, to facilitate cleaner and shorter representation of these steps in the later protocols.

8.11.2.1 Random secret sharing.

We introduce random secret sharing, in which a random secret is secret shared onto an output layer. This amounts to sampling a polynomial of degree at most t uniformly at random, independent of the view of the adversary, and distributing its shares onto the output layer. This primitive will function as a building block in implement the more complex functionalities we build in this section.

Random secret sharing functionality

PUBLIC PARAMETERS: No input layer, output layer $\mathcal{L}_d$, $d \geq 6$.
NOTATION: We will denote this functionality by $\langle s, f(x)\rangle_d \leftarrow \$$

Functionality samples $s \leftarrow \mathbb{F}$, and $c_l \leftarrow \mathbb{F}$ for each $1 \leq l \leq t$. Let $f(x) = s + \sum_{l=1}^{t} c_l x^l$. The functionality delivers $f(i)$ to $\mathcal{P}_i^d$ for each $i \in [n]$; i.e., distributes $\langle s, f(x)\rangle_d$

Figure 8.22: Random secret sharing functionality

Implementing random secret sharing. Using verifiable secret sharing, implementing this protocol is straight forward: We can take assistance from $t+1$ parties (from a previous layer) to verifiably secret share a random secret each onto the output layer. The parties take the sum of these shares as the sampled share. Since at least one amongst $t+1$ of the parties who supplied shares is honest, and parallel VSS is secure, the sum of these shares is guaranteed to be random independent of adversary's view. Given that our VSS protocol consumes 6 layers, random secret sharing with output layer d requires the parties in $d-6$ to supply random secrets. Hence, our resharing protocol works only when the output layer is $\mathcal{L}_6$ or later. This does not pose a limitation since random secret sharing is always used in $\mathcal{L}_6$ or later in all our implementations.

A t-secure protocol for random secret sharing

PUBLIC PARAMETERS: Output layer $\mathcal{L}_d$ for $d \geq 6$.
SUBROUTINES: A t-secure protocol $\Pi_{\text{ShamirVSS}}$ that implements $f_{\text{ShamirVSS}}$.

1. For each $i \in [t+1]$,
   a) $\mathcal{P}_i^{d-6}$ samples $s_i \leftarrow \mathbb{F}$ and $c_{i,l} \leftarrow \mathbb{F}$ for each $0 \leq l \leq t$, and defines $f_i(x) = s_i + \sum_{l=1}^{t} c_{i,l} x^l$.
   b) Execute $\Pi_{\text{ShamirVSS}}$ with $\mathcal{P}_i^{d-6}$ as dealer to verifiably secret share s_i using $f_i(x)$ onto $\mathcal{L}_d$.
2. Parties in $\mathcal{L}_d$ store $\langle \sum_{i=1}^{t+1} s_i, \sum_{i=1}^{t+1} f_i(x) \rangle_d = \sum_{i=1}^{t+1} \langle s_i, f_i(x) \rangle_d$.

Figure 8.23: A t-secure protocol for random secret sharing

The security of the protocol follows from the above discussion. In fact, the security of parallel invocations of the VSS protocol implies that parallel invocations of random secret sharing remain secure as well.

Theorem 8.14. *Protocol in Figure 8.23 realizes random secret sharing functionality in Figure 8.22 with perfect t-security for $t < n/3$.*

All the protocols we construct in the sequel use random secret shares in various ways. For simplicity, we will construct and analyze them assuming that the random secret shares are available as setup. To realize random secret sharing onto a layer using the protocol in Figure 8.23, VSS protocols need to be invoked with dealers situated 6 layers above the layer that requires the setup. For now, we overlook this fact and adhere to our convention of having the input client in $\mathcal{L}_0$ with random secret shares as setup made available whenever necessary. This is done to keep the descriptions simple; furthermore, all such protocols are constructed in order to be used as subroutines in the main protocol which implements efficient layered MPC for general function computation given in Figure 8.19. In our final construction, we replace the setup with concurrent protocols securely the setup and argue the security of the ensemble.

8.11.2.2 Resharing.

Going forward, in many protocols, we encounter scenarios where a value that has been verifiably secret shared in a layer needs to be replicated in a later layer. Naively duplicating the same secret sharing by sending the shares to the later layer using future messaging is clearly not secure. The adversary can corrupt t parties each in both the layers and learn $2t$ shares of the secret, breaking security. The later layer needs to necessarily receive a fresh resharing of the same value. The resharing functionality allows parties in $\mathcal{L}_a$ with (a valid) secret sharing of a secret s to "handover" the secret to parties in $\mathcal{L}_b$, for any $b > a$, by providing a fresh secret sharing of s. The following functionality requires that the input clients hold a valid Shamir secret sharing, and that at most t input clients are corrupt.

Resharing functionality

PUBLIC PARAMETERS: Output layer $\mathcal{L}_d$ for any $d \geq 1$.
SECRET INPUTS: $\langle s, f(x)\rangle_0$.
NOTATION: $\langle s, f'(x)\rangle_d \leftarrow \langle s, f(x)\rangle_0$.

1. Input clients reveal $\langle s, f(x)\rangle_0$ to the functionality who reconstructs s. The reconstruction is correct since at most t parties in $\mathcal{L}_0$ are corrupt.
2. Functionality samples $c_l \leftarrow \mathbb{F}, 1 \leq l \leq t$, defines $f'(x) = s + \sum_{i=1}^{t} c_l$, and distributes $\langle s, f'(x)\rangle_d$.

Figure 8.24: Resharing functionality

Implementing resharing. For implementing resharing, we use a setup in which t random secrets are secret shared onto the input clients. The protocol works as follows:

A t-secure implementation of resharing

PUBLIC PARAMETERS: Output layer $d \geq 1$.
SECRET INPUTS: $\langle s, f(x)\rangle_0$.
SETUP: $\langle \alpha_l, f_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$.
OUTPUT: $\langle s, f'(x)\rangle_d$, where $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$.

1. For each $i \in [n]$, execute

$$(s + \sum_{l=1}^{t} i^l \alpha_l)_{\mathcal{P}_i^d} \Leftarrow \langle s, f(x)\rangle_0 + \sum_{l=1}^{t} i^l \langle \alpha_l, f_l(x)\rangle_0$$

   Recall that this involves the following steps:
   a) For each $j \in [n]$, let $s_j = f(j)$ be the share of s held by $\mathcal{P}_j^0$. For each $1 \leq l \leq t$, let $\alpha_{l,j} = f_l(j)$ be the share of α_l held by $\mathcal{P}_j^0$. Then, $\mathcal{P}_j^0$ sends $s_j + \sum_{l=1}^{t} i^l \alpha_{l,j}$ to $\mathcal{P}_i^d$ using future messaging.
   b) For each $j \in [n]$, $\mathcal{P}_i^d$ recovers $s'_{i,j}$ as the output of future messaging with $\mathcal{P}_j^0$ as sender. $\mathcal{P}_i^d$ stores $\mathsf{Rec}(s'_{i,1}, \ldots, s'_{i,n})$ as their share of s.

      Whenever $\mathcal{P}_j^0$ is honest, $s_{i,j} = g(i)$, where $g(x) = f(x) + \sum_{l=1}^{t} i^l f_l(x)$. Since at most t parties are corrupt, $\mathcal{P}_j^0$ correctly recovers $s_i = g(0) = f'(i)$, where $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$.
2. Each $\mathcal{P}_i^d$ stores $s_i = s + \sum_{l=1}^{t} i^l \alpha_l$ as their (re)share of s.

Figure 8.25: A t-secure implementation of resharing

The correctness of the above protocol is clear from the description. Since $\langle \alpha_l, f_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$ are randomly sampled, by linearity of Shamir secret sharing and security of future messaging, the shares received by corrupt output clients are identically distributed irrespective of s and $f(x)$. Hence, the view of an adversary can be simulated even if it knows $s, f(x)$. Observe that, assuming the setup, the protocol necessarily implies par-

allel future messaging. Since our future messaging protocol is secure when executed in parallel, the resharing protocol also remains secure under parallel composition.

Theorem 8.15. *Protocol in Figure 8.25 realizes the resharing functionality in Figure 8.24 with perfect t-security for $t < n/3$.*

8.11.2.3 Reinforced secret sharing.

We next define an enhanced form of Shamir secret sharing that we will refer to as reinforced secret sharing. This notion is defined in [CDN15] as *verifiable secret sharing*, which we defined in differently. We will use reinforced secret shares for securely processing the gates during the circuit evaluation phase of our MPC protocol.

Definition 8.6. *A (t, n)-reinforced secret sharing of $s \in \mathbb{F}$ consists of the following $(n+1)$ distinct Shamir secret shares:*

1. *Sample a polynomial $f(x)$ of degree at most t uniformly at random under the constraint $f(0) = s$.*

2. *For each $i \in [n]$, sample a polynomial $f_i(x)$ of degree at most t uniformly at random under the constraint $f_i(0) = f(i)$.*

3. *Distribute shares $\langle s, f(x)\rangle$ and $\langle f(i), f_i(x)\rangle$.*

Reconstruction amounts to applying the reconstruction algorithm for Shamir secret sharing on shares $\langle s, f(x)\rangle$.

We will denote a reinforced secret sharing of a secret s using $f(x), (f_i(x))_{i\in[n]}$, as defined above, by $[[s, f(x), (f_i(x))_{i\in[n]}]]$.

In our constructions, we build a reinforced secret sharing of a secret from a valid Shamir secret sharing of the same. This notion is formalized by the reinforced resharing functionality described below. The functionality requires that the input clients hold a valid Shamir secret sharing, and that at most t input clients are corrupt.

Reinforced resharing functionality

PUBLIC PARAMETERS: Output layer $\mathcal{L}_d$ for any $d \geq 1$.
SECRET INPUTS: $\langle s, f(x)\rangle_0$.
NOTATION: $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d \leftarrow \langle s, f(x)\rangle_0$.

1. Input clients ($\mathcal{L}_0$) reveal $\langle s, f(x)\rangle_0$ to the functionality who recovers s. The reconstruction is correct since at most t parties are corrupt.
2. Functionality samples a random polynomial $f'(x)$ of degree at most t conditioned on $f'(0) = s$; for each $i \in [n]$, it samples a random polynomial $f'_i(x)$ of degree at most t conditioned on $f'_i(0) = f'(i)$.
3. Functionality distributes $\langle s, f'(x)\rangle_d$, and $\langle f'(i), f'_i(x)\rangle_d$ for each $i \in [n]$.

Figure 8.26: Reinforced resharing functionality

Implementing reinforced resharing. Our protocol works as follows: First, the secret s is reshared to the output clients. This involves sampling secret shares of random secrets α_l for each $1 \leq l \leq t$ and delivering $s_i = f(i)$ to output client i, where $f(x) = s + \sum_{l=1}^{t} \alpha_l x^l$. This is realized exactly as in our implementation of resharing (Figure 8.25). Observe that the input clients possess a secret sharing of s_i for each i as well, indeed, s_i was revealed to output client i by revealing the shares of s_i. But then, each s_i can be reshared onto the output layer using the resharing protocol. This achieves reinforced resharing of s. The security of the construction follows directly from the security of resharing. Similar to the resharing protocol, reinforced resharing only uses parallel invocations of future messaging protocol; hence, it remains secure under parallel composition.

A t-secure implementation of reinforced resharing

PUBLIC PARAMETERS: Output layer $d \geq 1$.
SECRET INPUTS: $\langle s, f(x)\rangle_0$.
SETUP: $\langle \alpha_l, g_l(x)\rangle_0 \leftarrow \$$ for $1 \leq l \leq t$ and $\langle \alpha_{i,l}, g_{i,l}(x)\rangle_0 \leftarrow \$$ for $i \in [n]$ and $1 \leq l \leq t$.
SUBROUTINES: A t-secure protocol Π_{reshare} that implements the resharing functionality.
OUTPUT: $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d$.

1. For each $i \in [n]$, define

$$\langle s_i, g'_i(x)\rangle_0 = \langle s, f(x)\rangle_0 + \sum_{l=1}^{t} i^l \langle \alpha_l, g_l(x)\rangle_0.$$

   Here, $f'(x) = s + \sum_{l=1}^{t} \alpha_l x^l$ and $s_i = f'(i)$ for all $i \in [n]$.
2. For each $i \in [n]$, execute $(s_i)_{\mathcal{P}_i^d} \Leftarrow \langle s_i, g'_i(x)\rangle_0$ (See Figure 8.24 step 1).
3. For each $i \in [n]$, reshare $\langle s_i, g'_i(x)\rangle_0$ using Π_{reshare} using $\langle \alpha_{i,l}, g_{i,l}(x)\rangle_0$ for $1 \leq l \leq t$ as setup. For each $i \in [n]$, this achieves

$$\langle s_i, f'_i(x)\rangle_d \leftarrow \langle s_i, g'_i(x)\rangle_0, \text{ where } f'_i(x) = f'(i) + \alpha_{i,1}x^1 + \ldots + .\alpha_{i,t}x^t.$$

4. Parties in $\mathcal{L}_d$ store $[[s, f'(x), (f'_i(x))_{i\in[n]}]]_d$.

Figure 8.27: A t-secure implementation of reinforced resharing

Theorem 8.16. *Protocol in Figure 8.25 realizes the reinforced resharing functionality in Figure 8.24 with perfect t-security for $t < n/3$.*

8.11.3 Details omitted from Section 8.5.2

We define the functionality for multiplication with helper in Figure 8.28.

Multiplication with helper functionality

PUBLIC PARAMETERS: Helper is $\mathcal{P}_1^0$ and output layer $\mathcal{L}_9$.
INPUT: $\langle \alpha, f_0(x) \rangle_0$, $\langle \beta, g_0(x) \rangle_0$; helper $\mathcal{P}_1^0$ holds α, β.

1. Input clients ($\mathcal{L}_0$) reveal the shares of α and β to the functionality, who reconstructs α, β. Additionally, $\mathcal{P}_1^0$ sends γ to the functionality.
2. If $\gamma = \alpha\beta$, functionality distributes a secret sharing of γ onto $\mathcal{L}_3$; i.e., $\langle \alpha\beta, h(x) \rangle_9$, where $h(x)$ is a random polynomial of degree at most t conditioned on $h(0) = \alpha\beta$. Otherwise, functionality delivers $\perp$ to all parties.

Figure 8.28: Multiplication with helper functionality

The protocol for multiplication with helper is formally described in Figure 8.29.

t-securely realizing multiplication with helper

PUBLIC PARAMETERS: Input layer $\mathcal{L}_0$, helper $\mathcal{P}_1^0$, output layer $\mathcal{L}_8$.
INPUTS: $\langle \alpha, f_0(x)\rangle_0, \langle \beta, g_0(x)\rangle_0$; helper $\mathcal{P}_1^0$ holds α, β.
SUBROUTINES: $\Pi_{\text{ShamirVSS}}$ and Π_{Reshare} implementing verifiable secret sharing and resharing functionality.
SETUP: Sufficiently many random secret shares in $\mathcal{L}_0$ and $\mathcal{L}_6$ required to execute resharing.

1. $\mathcal{P}_1^0$ samples $\alpha_l \leftarrow \mathbb{F}$, $\beta_l \leftarrow \mathbb{F}$ for each $1 \le l \le t$. Define $f(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l$ and $g(x) = \beta + \sum_{l=1}^{t} \beta_l x^l$. Let $f(x)g(x) = \sum_{l=0}^{2t} \gamma_l x^l$. Use $\Pi_{\text{ShamirVSS}}$ with $\mathcal{P}_1^0$ as dealer and $\mathcal{L}_6$ as shareholders, to distribute shares

$$\langle \alpha_l, f_l(x)\rangle_6, \forall 1 \le l \le t \quad \langle \beta_l, g_l(x)\rangle_6, \forall 1 \le l \le t \quad \langle \gamma_l, h_l(x)\rangle_6, \forall 0 \le l \le 2t$$

   Finally, execute Π_{Reshare} to realize

$$\langle \alpha, f_0'(x)\rangle_6 \leftarrow \langle \alpha, f_0(x)\rangle_0 \qquad \langle \beta, g_0'(x)\rangle_6 \leftarrow \langle \beta, g_0(x)\rangle_0$$

2. For each $i \in [n]$, reveal the following linear combinations of shares to $\mathcal{P}_i^7$:

$$(\hat{f}(i))_{\mathcal{P}_i^7} \Leftarrow \langle \alpha, f_0'(x)\rangle_6 + i\langle \alpha_1, f_1(x)\rangle_6 + \ldots + i^t\langle \alpha_t, f_t(x)\rangle_6$$
$$(\hat{g}(i))_{\mathcal{P}_i^7} \Leftarrow \langle \beta, g_0'(x)\rangle_6 + i\langle \beta_1, g_1(x)\rangle_6 + \ldots + i^t\langle \alpha_t, f_t(x)\rangle_6$$
$$(\hat{h}(i))_{\mathcal{P}_i^7} \Leftarrow \langle \gamma_0, h_0'(x)\rangle_6 + i\langle \gamma_1, h_1(x)\rangle_6 + \ldots + i^{2t}\langle \gamma_{2t}, h_t(x)\rangle_6$$

   Here, $\hat{f}(x) = \alpha + \sum_{l=1}^{t} \alpha_l x^l$, $\hat{g}(x) = \beta + \sum_{l=1}^{t} \beta_l x^l$ and $\hat{h}(x) = \sum_{l=0}^{2t} \gamma_l x^l$. Further, execute Π_{Reshare} to realize $\langle \gamma_0, h(x)\rangle_9 \leftarrow \langle \gamma_0, h_0\rangle_6$.

3. Each $\mathcal{P}_i^7, i \in [n]$ checks if $\hat{f}(i)\hat{g}(i) = \hat{h}(i)$. Otherwise, broadcast a complaint.

4. For each $i \in [n]$, if $\mathcal{P}_i^7$ registered a complaint, execute public reveal as follows:

$$(\hat{f}(i))_9 \Leftarrow \langle \alpha, f_0'(x)\rangle_6 + i\langle \alpha_1, f_1(x)\rangle_6 + \ldots + i^t\langle \alpha_t, f_t(x)\rangle_6$$
$$(\hat{g}(i))_9 \Leftarrow \langle \beta, g_0'(x)\rangle_6 + i\langle \beta_1, g_1(x)\rangle_6 + \ldots + i^t\langle \alpha_t, f_t(x)\rangle_6$$
$$(\hat{h}(i))_9 \Leftarrow \langle \gamma_0, h_0'(x)\rangle_6 + i\langle \gamma_1, h_1(x)\rangle_6 + \ldots + i^{2t}\langle \gamma_{2t}, h_t(x)\rangle_6$$

   Note that, $\hat{f}(i), \hat{g}(i)$ and $\hat{h}(i)$ are to be revealed only for $i \in [n]$ with a registered complaint. Since the complaints are available in $\mathcal{L}_8$, this can be achieved by having each $\mathcal{P}_i^6$ secret share their share of $\hat{f}(i)$ and so on, onto $\mathcal{L}_8$ and then having $\mathcal{L}_8$ selectively reveal these shares to $\mathcal{P}_i^9$ only for $i \in [n]$ with a complaint.

5. For each $i \in [n]$ with a complaint, all parties in $\mathcal{L}_9$ check if $\hat{f}(i)\hat{g}(i) = \hat{h}(i)$ If the equality check succeeds for all complaints, then the parties store $\langle \gamma_0, h(x)\rangle_9$ as shares of $\alpha\beta$.

Figure 8.29: t-securely realizing multiplication with helper

Part IV

Epilogue

Chapter 9

Conclusion

9.1 Discussion

This thesis explores a broad spectrum of problems but all are within the area of research of MPC with dynamic groups. Inspired by the elegant separation of concerns suggested in [GHK+21], we believe that (as in the standard model) one should be able to decouple the actual protocol from the infrastructure it is built on. This thesis reflects that. Encryption to the Future is an attempt to improve on the underlying infrastructure in this domain not only by suggesting more efficient (non-interactive) protocols but also by putting the focus on the bare-bone piece of infrastructure that is needed to communicate in this setting—much like point-to-point channels play an important role in existing standard MPC.

The seminal work of [BGG+20] was arguably the first to (re-)open this area of research and showed a simple protocol for a "blockchain to keep a secret". This work allowed a randomly selected committee to share the secret to the next committee while each committee member speaks only once. However, the proof of correct re-sharing—that the shares are valid and that they correspond to public ciphertext that holds an earlier share—was taken care of by use of expensive generic NIZKs. Despite the popularity of this line of research, no follow-up work addressed this inefficiency even when it took on new importance with the advent of YOSO MPC where re-sharing is even more ubiquitous. YOLO YOSO is the first work to address this issue by proposing highly optimized proof of correct resharing which is compatible with the existing communication infrastructure.

The last contribution, Layered MPC, sprung out of a need to unite the research of MPC with dynamic groups with the existing literature on proactive MPC. Moreover, we discovered that the problem of perfect MPC with G.O.D. and optimal corruption threshold remained open in both areas of research. We observed that non of the existing models on MPC with dynamic groups supported our investigation *i.e.* protocols in these models did not exhibit the necessary implications for the proactive model. The design of Layered MPC is an attempt to distill the essence of MPC with dynamic groups but can be relaxed in ways that mimics other models [CGG+21, GHK+21, AHKP22]. Layered MPC supports the recurring theme in this thesis of decoupling protocols and underlying infrastructure. The model assumes secure point-to-point (and broadcast) channels between nodes forming a layered communication network. Since the model is really just standard MPC with restricted interaction pattern, protocols in this model can be analyzed within well-established se-

curity frameworks such as UC. Finally, it has interesting implications for a much older area of cryptographic research, namely, proactive MPC [OY91].

Permissionless blockchains have given new life to the research area of MPC with dynamic groups. Earlier, the main motivation was resilience towards a powerful adversary but now the dynamism also enables support for a high churn rate among nodes and allows for even a small resource commitment from a node to contribute to the overall computation. As a testament to this fact one need only to observe the number of new models that, within the last few years, have been designed to support this area of research; Fluid MPC [CGG+21], YOSO MPC [GHK+21], SCALES [AHKP22] and now Layered MPC [DKI+23]. Why is it so hard to agree on a model for this area? One reason might be that the research on permissionless blockchains is itself in active development so all the models above are basically built on shifting sand. Another factor could be that modelling is just incredibly challenging and that it might take decades to converge toward a coherent model that suit most protocol designers. After all, this is what happened with the standard model of cryptography [GL91, Bea92b, MR92, Can00, Can01].

9.2 Future Work

Below, we offer a list of future work and interesting areas to explore in the realm of MPC with dynamic committees.

- Statistical Layered MPC: In the work on layered MPC we propose a protocol for perfect security with G.O.D. and tolerated corruption threshold of $t < n/3$. Moreover, we sketch a protocol that achieve computational security (relying only on homomorphic commitments) with tolerated corruption threshold of $t < n/2$. What remains for a full characterization of feasibility in Layered MPC is protocol that captures the feasibility in the statistical setting. Interestingly, one of the first protocols in the YOSO model [GHK+21] proved secure in the statistical setting. However, this protocol are not directly "portable" to the layered MPC setting due to absence of ideal target-anonymous channels to arbitrary future committee members. It is interesting to revisit this problem of statistical Layered MPC and investigate the possibility of achieving security in an honest majority setting.

- The power of Encryption to the Future: Due to the probabilistic nature of role assignment in YOSO MPC no protocol can obtain optimal threshold with constant t and n (even assuming large committee sizes). In the effort of transforming existing secure YOSO MPC protocols to secure Layered MPC protocols, it became clear that this is not the only thing that separates the two models. The ability to send messages to committee members (and have the sender being committed to this message) in a larger future horizon is another big differentiator. From the work on EtF we already have an idea about the strong assumptions necessary to construct the primitive that support "far-future" EtF. However, this was only studied at the level of the primitive and not in the context of protocols using EtF. It is interesting to explore and quantify the power that ideal channels to (far)-future committee members provide to protocols that use them.

- Forward Secure Encryption to the Future: In the YOSO model it is assumed that when a party has spoken on behalf of its role, it will then never speak again. But even in a blockchain setting nodes may stay around for longer periods of time and want to contribute in more than just a single round. This idea is also supported by the work of Fluid MPC setting which allows parties to commit to more than a single round. The communication in these models (including EtF) is constructed by allowing ciphertexts to be posted on a PoS blockchain ledger. However, this introduces a grave issue pertaining to forward security: If a party is corrupted at some point in the future, the adversary can readily access and decrypt all ciphertexts for which this party was elected as receiver in the past (including all of those in the future). It is interesting to research the possibility of leader election using key-evolving VRFs and let that define the specific lottery predicate in the EtF construction. Potentially, this could provide a useful combination of EtF with key-evolving properties.

- Unification of dynamic Models: Apart from the usual suspects of models of MPC based on dynamic committees [CGG+21, GHK+21] and corresponding derivative models [AHKP22, BBG+21, DGLZ23, BEP23, RS22], there is a large unexplored territory of interesting models that also favours dynamism in protocols [DER21, GBO+23, GPS19] but captures the dynamic properties in a slightly different way. It would be interesting (albeit ambitious) and beneficial to this area of research to take steps to unify all of these models focusing on dynamism.

Bibliography

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Heidelberg, May / June 2006.

[AHKP22] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 502–531. Springer, Heidelberg, November 2022.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *J. Cryptol.*, 30(1):58–151, jan 2017.

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[BBG+21] Fabrice Benhamouda, Elette Boyle, Niv Gilboa, Shai Halevi, Yuval Ishai, and Ariel Nof. Generalized pseudorandom secret sharing and efficient straggler-resilient secure computation. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 129–161. Springer, Heidelberg, November 2021.

[BBH06] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 226–243. Springer, Heidelberg, February 2006.

[BCD+08] Peter Bogetoft, Dan Lund Christensen, Ivan Damgard, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. https://eprint.iacr.org/2008/068.

[BDH+19] Michael Backes, Nico Döttling, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. Ring signatures: Logarithmic-size, no setup - from standard as-

sumptions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 281–311. Springer, Heidelberg, May 2019.

[BDO22] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. Cryptology ePrint Archive, Report 2022/1437, 2022. https://eprint.iacr.org/2022/1437.

[Bea92a] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.

[Bea92b] Donald Beaver. Foundations of secure interactive computing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 377–391. Springer, Heidelberg, August 1992.

[BEHG20] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. Cryptology ePrint Archive, Report 2020/025, 2020. https://eprint.iacr.org/2020/025.

[BELO15] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. Communication-optimal proactive secret sharing for dynamic groups. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 23–41. Springer, Heidelberg, June 2015.

[BEP23] Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid mpc. Cryptology ePrint Archive, Paper 2023/839, 2023. https://eprint.iacr.org/2023/839.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[BGG+20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018*, volume 94, pages 21:1–21:21. LIPIcs, January 2018.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

[BIOW20] Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J. Wu. On succinct arguments and witness encryption from groups. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 776–806. Springer, Heidelberg, August 2020.

[BKLZL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Heidelberg, November 2020.

[BKRS18] Elette Boyle, Saleet Klein, Alon Rosen, and Gil Segev. Securing abe's mix-net against malicious verifiers via witness indistinguishability. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 274–291. Springer, Heidelberg, September 2018.

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg, November 2020.

[Bra85] Gabriel Bracha. An $O(\lg n)$ expected rounds randomized byzantine generals protocol. In *17th ACM STOC*, pages 316–326. ACM Press, May 1985.

[BT99] Fabrice Boudot and Jacques Traoré. Efficient publicly verifiable secret sharing schemes with fast or delayed recovery. In Vijay Varadharajan and Yi Mu, editors, *ICICS 99*, volume 1726 of *LNCS*, pages 87–102. Springer, Heidelberg, November 1999.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CD17] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556. Springer, Heidelberg, July 2017.

[CD20] Ignacio Cascudo and Bernardo David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2020.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Heidelberg, May 1999.

[CDD+04] Ran Canetti, Ivan Damgard, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. Adaptive versus non-adaptive security of multi-party protocols. *Journal of Cryptology*, 17:153–207, 2004.

[CDGK22] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 651–680. Springer, Heidelberg, December 2022.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 342–362. Springer, Heidelberg, February 2005.

[CDK+22] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 151–180. Springer, Heidelberg, December 2022.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 234–264. Springer, Heidelberg, March 2016.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CFG22] Dario Catalano, Dario Fiore, and Emanuele Giunta. Adaptively secure single secret leader election from DDH. Cryptology ePrint Archive, Report 2022/687, 2022. https://eprint.iacr.org/2022/687.

[CGG+21] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, August 2021. Springer, Heidelberg.

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 719–728. ACM Press, October / November 2017.

[CH01] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *Advances in Cryptology—CRYPTO'94: 14th Annual International Cryptology Conference Santa Barbara, California, USA August 21–25, 1994 Proceedings*, pages 425–438. Springer, 2001.

[CKL06] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, April 2006.

[CKLS02] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for

adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, April / May 2002.

[DEP21] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. Cryptology ePrint Archive, Paper 2021/1376, 2021.

[DER21] Ivan Damgård, Daniel Escudero, and Divya Ravi. Information-theoretically secure MPC against mixed dynamic adversaries. Cryptology ePrint Archive, Report 2021/1163, 2021. https://eprint.iacr.org/2021/1163.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.

[DGLZ23] Giovanni Deligios, Aarushi Goel, and Chen-Da Liu-Zhang. Maximally-fluid mpc with guaranteed output delivery. Cryptology ePrint Archive, Paper 2023/415, 2023.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 501–520. Springer, Heidelberg, August 2006.

[DJ97] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications. Technical report, Citeseer, 1997.

[DKI+23] Bernardo David, Anders Konring, Yuval Ishai, Eyal Kushilevitz, and Varun Narayanan. Perfect MPC over layered graphs. Cryptology ePrint Archive, Report 2023/330, 2023. https://eprint.iacr.org/2023/330.

[DLS84] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony (preliminary version). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 103–118. ACM, August 1984.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Heidelberg, February 2019.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[DS15] David Derler and Daniel Slamanig. Practical witness encryption for algebraic languages and how to reply an unknown whistleblower. Cryptology ePrint Archive, Report 2015/1073, 2015. https://eprint.iacr.org/2015/1073.

[ELL20] Karim Eldefrawy, Tancrède Lepoint, and Antonin Leroux. Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20, Part I*, volume 12146 of *LNCS*, pages 3–23. Springer, Heidelberg, October 2020.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In Elizabeth Borowsky and Sergio Rajsbaum, editors, *22nd ACM PODC*, pages 211–220. ACM, July 2003.

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 121–136. Springer, Heidelberg, August 1998.

[FL81] Michael J Fischer and Nancy A Lynch. A lower bound for the time to assure interactive consistency. Technical report, GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION AND COMPUTER SCIENCE, 1981.

[FLM85] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In Michael A. Malcolm and H. Raymond Strong, editors, *4th ACM PODC*, pages 59–70. ACM, August 1985.

[FLM86] Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1:26–39, 1986.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[FLZL21] Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 355–362, 2021.

[FM85] Paul Feldman and Silvio Micali. Byzantine agreement in constant expected time (and trusting no one). In *26th FOCS*, pages 267–276. IEEE Computer Society Press, October 1985.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

[FO98] Eiichiro Fujisaki and Tatsuaki Okamoto. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In Kaisa

Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 32–46. Springer, Heidelberg, May / June 1998.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

[Gar94] Juan A Garay. Reaching (and maintaining) agreement in the presence of mobile faults. In *International Workshop on Distributed Algorithms*, pages 253–264. Springer, 1994.

[GBO+23] Mariana Gama, Emad Heydari Beni, Emmanuela Orsini, Nigel P. Smart, and Oliver Zajonc. MPC with delayed parties over star-like networks. Cryptology ePrint Archive, Report 2023/096, 2023. https://eprint.iacr.org/2023/096.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 529–561. Springer, Heidelberg, November 2017.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGJ+15] Juan A. Garay, Ran Gelles, David S. Johnson, Aggelos Kiayias, and Moti Yung. A little honesty goes a long way - the two-tier model for secure multiparty computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 134–158. Springer, Heidelberg, March 2015.

[GGLZ22] Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 96–119. Springer, Heidelberg, May / June 2022.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure

MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

[GHL21] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. Cryptology ePrint Archive, Report 2021/1397, 2021. https://eprint.iacr.org/2021/1397.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[GHM+21] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 32–61. Springer, Heidelberg, November 2021.

[GIKR01] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*, pages 580–589. ACM Press, July 2001.

[GIOZ17] Juan A. Garay, Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. The price of low communication in secure multi-party computation. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 420–446. Springer, Heidelberg, August 2017.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part I*, volume 13177 of *LNCS*, pages 252–282. Springer, Heidelberg, March 2022.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 77–93. Springer, Heidelberg, August 1991.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443. Springer, Heidelberg, August 2014.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental

game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GNQT20] Lydia Garms, Siaw-Lynn Ng, Elizabeth A. Quaglia, and Giulia Traverso. Anonymity and rewards in peer rating systems. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 277–297. Springer, Heidelberg, September 2020.

[Gol09] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[GPS19] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019.

[HIJ+16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *ITCS 2016*, pages 157–168. ACM, January 2016.

[HJKY95] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Heidelberg, August 1995.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptol.*, 13(1):31–60, 2000.

[HV09] Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 294–308. Springer, Heidelberg, August 2009.

[ISN89] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 955–966. ACM Press, November 2013.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.

[KLR06] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 109–118. ACM Press, May 2006.

[KLR10] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. *SIAM Journal on Computing*, 39(5):2090–2112, 2010.

[KRY22] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Towards efficient YOSO MPC without setup. Cryptology ePrint Archive, Report 2022/187, 2022. https://eprint.iacr.org/2022/187.

[LAM98] LESLIE LAMPORT. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[Lin03] Yehuda Lindell. General composition and universal composability in secure multi-party computation. In *44th FOCS*, pages 394–403. IEEE Computer Society Press, October 2003.

[Lin17] Yehuda Lindell. Fast secure two-party ECDSA signing. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Heidelberg, August 2017.

[LRSW99] Anna Lysyanskaya, Ronald L. Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard M. Heys and Carlisle M. Adams, editors, *SAC 1999*, volume 1758 of *LNCS*, pages 184–199. Springer, Heidelberg, August 1999.

[LSP82] LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 325–335. Springer, Heidelberg, July 2004.

[Mau06] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[Mic17] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

[MR92] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 392–404. Springer, Heidelberg, August 1992.

[MZW+19] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2369–2386. ACM Press, November 2019.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, 2008.

[Nie03] Jesper Buus Nielsen. *On protocol security in the cryptographic model*. Citeseer, 2003.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[PSs17] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, April / May 2017.

[RA23] Flashbots Robert Annessi. Backrunning private transactions using multi-party computation, 2023. https://writings.flashbots.net/backrunning-private-txs-MPC/, Last accessed on 2023-08-26.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Heidelberg, August 2022.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565. Springer, Heidelberg, December 2001.

[RV05] A. Ruiz and J. L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In *WEWoRC 2005–Western European Workshop on Research in Cryptology*, 2005.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

[Sch99] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164. Springer, Heidelberg, August 1999.

[SLL10] David Schultz, Barbara Liskov, and Moses Liskov. Mpss: mobile proactive secret sharing. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–32, 2010.

[Sta96] Markus Stadler. Publicly verifiable secret sharing. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 190–199. Springer, Heidelberg, May 1996.

[W+14] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WWW02] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. Verifiable secret redistribution for archive systems. In *First International IEEE Security in Storage Workshop, 2002. Proceedings.*, pages 94–105. IEEE, 2002.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[YMR+19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 347–356. ACM, July / August 2019.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.