Expanding Blockchain Horizons through Privacy-Preserving Computation

Lorenzo Gentile

PhD thesis
IT University of Copenhagen
2023
Computer Science Department

IT UNIVERSITY OF COPENHAGEN

Summary in English

In this thesis, we explore the area lying between privacy-preserving computation and blockchain applications. In particular, we consider the following applications: auctions, *i.e.*, how to efficiently run auctions without an auctioneer while keeping the bids private; decentralized finance (DeFi), *i.e.*, what are the current solutions and the open problems related to front-running; anonymous credentials, *i.e.*, how to find a trade-off between privacy and accountability of the users.

In the context of auctions, we consider specifically the case of sealed-bid auctions, *i.e.*, no bidder is supposed to know how much the other auction participants have bid. Sealed-bid auctions are a common way of allocating an asset among a set of parties but require trusting an auctioneer who analyses the bids and determines the winner. Many privacy-preserving computation protocols for auctions have been proposed to eliminate the need for a trusted third party. However, they lack *fairness*, meaning that the adversary learns the outcome of the auction before honest parties and may choose to make the protocol fail without suffering any consequences. In this thesis, we propose efficient protocols for both first and second-price sealed-bid auctions with fairness against rational adversaries, leveraging *secret* cryptocurrency transactions and public smart contracts. In our approach, the bidders jointly compute the winner of the auction while preserving the privacy of losing bids and ensuring that cheaters are financially punished by losing a *secret* collateral deposit. We guarantee that it is never profitable for rational adversaries to cheat by making the deposit equal to the bid plus the cost of running the protocol, *i.e.*, once a party commits to a bid, it is guaranteed that it has the funds and it cannot walk away from the protocol without forfeiting the bid. Moreover, our protocols ensure that the winner is determined and the auction payments are completed even if the adversary misbehaves so that it cannot force the protocol to fail and then rejoin the auction with an adjusted bid. In comparison to the state-of-the-art, our constructions are both more efficient and furthermore achieve stronger security properties, *i.e.*, fairness. Interestingly, we show how the second-price can be computed with a minimal increase of the complexity of the simpler first-price case. Moreover, in case there is no cheating, only collateral deposit and refund transactions must be sent to the smart contract, significantly saving on-chain storage.

In the context of decentralized finance, we take into consideration the prob-

lem of front-running. Front-running is the malicious, and often illegal, act of both manipulating the order of pending trades and injecting additional trades to make a profit at the cost of other users. In decentralized finance, front-running strategies exploit both public knowledge of user trades from transactions pending on the network and the miner's ability to determine the final transaction order. Given the financial loss and increased transaction load resulting from adversarial front-running in decentralized finance, novel cryptographic protocols have been proposed to mitigate such attacks in the permission-less blockchain setting. In this thesis, we systematize and discuss the state-of-the-art of front-running mitigation in decentralized finance, and illustrate remaining attacks and open challenges.

Finally, in the context of anonymous credentials, we study the notion of anonymous credentials with *Publicly Auditable Privacy Revocation* (PAPR). PAPR credentials simultaneously provide *conditional* user privacy and *auditable* privacy revocation. The first property implies that users keep their identity private when authenticating unless and until an appointed authority requests to revoke this privacy, retroactively. The second property enforces that the auditors can verify whether or not this authority has revoked privacy from an issued credential (*i.e.*, learned the identity of the user who owns that credential), holding the authority accountable. In other words, the second property enriches anonymous credential systems with transparency by design, effectively discouraging such systems from being used for mass surveillance. In this thesis, we introduce the notion of a PAPR anonymous credential scheme, formalize it as an ideal functionality, and present constructions that are provably secure under standard assumptions in the Universal Composability framework. The core tool in our PAPR construction is a mechanism for randomly selecting an anonymous committee towards which users secret share their identity information, while hiding the identities of the committee members from the authority. As a consequence, in order to initiate the revocation process for a given credential, the authority is forced to post a request on a public bulletin board used as a broadcast channel to contact the anonymous committee that holds shares of the identity connected to the credential. This mechanism makes the user de-anonymization publicly auditable. Finally, we show how to modify our construction to obtain proactive security.

Overall, the goal of this thesis is to contribute to the extension and enhancement of blockchain applications through the usage of privacy-preserving computation.

Summary in Danish

Målet for denne opgave er at udforske området mellem teknologien for distribuerede private beregninger og blockchain applikationer. Opgaven behandler især følgende applikationer: auktioner - hvordan man kan skabe auktioner uden en tredje-part der spiller rollen som auktionarius mens deltagerne samtidig holder deres bud hemmelige. Decentralized Finance (DeFi) - her studeres de løsninger og åbne problemer som dette områder tilbyder specielt relateret til algoritmehandel og ubalance i markedsinformation. Anonyme personlige oplysninger - hvor omdrejningspunktet er hvordan man finder balancen mellem privatliv og brugernes ansvarlighed.

Vi fokuserer på auktioner hvor buddende i første omgang er forsejlede altså hvor ingen af deltagerne kender hinandens bud. Dette er en velkendt auktionsform hvor en betroet tredje-part modtager genstanden for auktionen samt hemmelige bud fra hver deltager og derefter udråber en vinder. Gennem tiden er der forelået mange protokoller for privat beregning af auktioner it uden en betroet tredje-part. Fælles for dem alle er at de ikke er fuldstændigt retfærdige. Det bevirker at en ondsindet deltager kan modtage resultatet af auktionen og derefter anullere protokollen uden følger. I denne opgave presenterer vi hurtige og effektive protokoller for både højeste- og næsthøjeste bud auktioner. Ved at udnytte kryptovaluta og smarte kontrakter kan vi garantere at disse protokoller giver en retfærdig auktion hvis alle deltagere handler rationelt. Vi bruger en teknik hvor deltagerne ved hjælp af en protokol kan beregne vinderen af auktionen mens alle andre bud forbliver hemmelige. Deltagere der forsøger at snyde bliver finansielt straffet da de mister et beløb som er blevet stillet i sikkerhed. Vi garanterer at det aldrig er profitabelt for en deltager at snyde da det beløb der bliver stillet i sikkerhed er det samme som buddet plus omkostninger for protokollen. Det vil sige at når først deltageren har budt så kan denne ikke afbryde deltagelsen i auktionen uden af miste sine penge. Ydermere, vores protokoller er designet så en vinder altid bliver udråbt og bud og betalinger er endelige selvom nogle deltagere skulle prøve at snyde ved at forsøge at anullere auktionen eller lave om på deres bud. Hvis vi sammenligner vores protokol med den nyeste forskning så er effekten ved vores design både en hurtigere og mere retfærdig auktion. En interesant konsekvens af vores design er at næsthøjeste bud auktioner kan beregnes med en meget lille øgning i kompleksitet sammenlignet med højeste bud auktioner. Hvis ingen deltagere forsøger at snyde så

behøver den smarte kontrakt kun at behandle sikkerhedstillelse og refundering hvilket giver en besparelse på den totale mængde information som protokollen gør brug af på den underliggende blockchain.

I området Decentralized Finance (DeFi) kigger vi især "Front-running". "Front-running" er en, ofte, ulovlig måde at drive algoritmehandel på hvor en kriminel part kan manipulere handler der er undervejs ved at omarrangere dem eller indlægge små, hyggpige handler før og efter der, når de finansielle transaktioner er afsluttet, skaber et fordelagtigt udfald for den kriminelle part med den almindelig DeFi bruger som offer. I DeFi bruger "Front-running" strategier både information fra finansielle transkaktioner der er undervejs på netværket men også en "miners" evne til at manipulere rækkefølgen og tilføje andre transaktioner inden de bliver afsluttet. Dette finansielle tab for brugeren sammenholdt med den stigende transaktionsbelastning hos mange blockchains giver anledning til at søge muligheder i kryptografisk protokoller for afhjælpe dette problem. I denne opgave sætter vi den nuværende viden i system og diskuterer måder hvorpå man kan afhjælpe "Front-running" samt andre skadelige strategier og fremtidige udfordringer.

Til sidst, omkring emnet anonyme personlige oplysninger, studerer vi muligheden for revision ved samtidig brug af anonyme personlige oplysninger *Publicly Auditable Privacy Revocation* (PAPR). Udfordringen er at man både vil tilbyde brugeren at være privat men samtidig, under specielle forhold, vil være i stand til at tilbagekalde dette. Det udmunder i et design hvor brugerens identitet er privat under brugen af systemet indtil en autoritet laver en forpørgsel for at få fjernet denne anonymitet. Hvis brugeren kommer under revision kan det nu verificeres at autoriteten har handlet korrekt og de-anonymiseret den rette identitet som brugeren har. Sidstnævnte er vigtigt for at skabe transparens og forebygge et system med masseovervågning. Vi introducerer PAPR som et koncept for anonyme personlige oplysninger. Vi formulerer dette som en ideal funktionalitet og presenterer protokoller som er beviseligt sikrer under normale antagelser i det såkaldte UC framework. Det vigtigste værktøj i PAPR er en mekanisme som tilfældigt vælger anonyme kommiteer som, til sammen, holder de private identitetsinformationer for alle brugere af systemet. En konsekvens af dette er at hvis en autoritet skal fjerne anonymiteten hos en bruger skal autoriteten sende en offentlig forespørgsel for at kontakte kommiteen som holder information om den forespurgte identitet. Det gør så at de-anonymiseringen kun kan gøres offentligt og skaber den rette transparens under revision. Vi afslutter med at vise hvordan vores konstruktion kan blive modificeret til at skabe proaktiv sikkerhed.

Overordnet set er målet med denne afhandling at bidrage til udvidelsen og forbedringen af blockchain-applikationer gennem brug af databehandling, der beskytter privatlivets fred.

Acknowledgements

Several people have contributed in their own way to the completion of this work. First of all, I am grateful to my supervisor Bernardo David for guiding me throughout my PhD studies. A passionate mentor who allowed me to develop my knowledge in the field of cryptography and to integrate easily into the scientific community around it. My appreciation as a mentor has to be extended to Sebastian Faust, who guided me during my exchange abroad.

Furthermore, my gratitude is due to my coauthors Bernardo David, Mohsen Pourpouneh, Carsten Baum, James Hsin-yu Chiang, Tore Kasper Frederiksen, Joakim Brorsson, Elena Pagnin, Paul Stankovski Wagner and my colleagues Ravi Kishore, Anders Konring, Felix Engelmann, Ieva Daukantas, Boel Nelson, Daniele Friolo, Gennaro Avitabile, Paola de Perthuis, Mariana Botelho da Gama, Anca Nitulescu, Orfeas Stefanos Thyfronitis Litos, Elena Micheli, Francesco Berti, Patrick Harasser and each member of the Center for Information Security and Trust (CISAT), as well as the whole academic and administrative staff involved in the process. Sharing the joy and the pain of research with them has been a precious professional and human experience. Beyond that, I acknowledge Concordium Foundation for supporting my work.

In addition to all this, I wish to especially thank my friends Eloisa, Linda, Massimo, Orfeas, Gianluca, Miriam, Mihhail, Mohamed, Ivan, Jon, Nikki, Anna, Daniele, Mario, Sumero, Celeste and Giada for their invaluable closeness during this journey. By all means, I extend my gratitude as friends to multiple people mentioned earlier, the crew I spent my days with during my exchange in Darmstadt, as well as others not explicitly mentioned here.

Finally, I express my deep gratitude to my family for cheering on me and to my partner Elena, in particular for still being my partner despite the occasionally unpredictable lifestyle I had during my PhD studies.

The inferno of the living is not something that will be; if there is one, it is what is already here, the inferno where we live every day, that we form by being together. There are two ways to escape suffering it. The first is easy for many: accept the inferno and become such a part of it that you can no longer see it. The second is risky and demands constant vigilance and apprehension: seek and learn to recognize who and what, in the midst of inferno, are not inferno, then make them endure, give them space.

— Italo Calvino, *Invisible Cities*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Privacy-preserving computation (also referred to as secure multi-party computation, secure computation, multi-party computation or MPC) is a subfield of cryptography with the goal of allowing a set of mutually distrusting parties to evaluate a certain function while keeping their input secret to each other. It traces its roots back to the 1980s, with works related to specific applications such as Mental Poker [163], investigating how two potentially dishonest players can play a fair game of Poker without using any cards, *e.g.*, over a phone, and the Millionaires' problem [174], investigating how two millionaires can learn which of them is richer without revealing their actual wealth. Both scenarios show how cryptography can be used to execute different computational tasks without requiring a trusted third party. Indeed, it has been proven that *general purpose MPC* is possible, *i.e.*, any function can be evaluated on private inputs [176].

Likewise, the blockchain was introduced through a specific application as well in 2008, *i.e.*, Bitcoin [150], which is a payment system based on a public transaction ledger, stored in a distributed way in a data structure composed by a list of implicitly ordered blocks, *i.e.*, a blockchain, and maintained by parties called *miners*, who have the chance to generate new blocks containing transactions by solving a puzzle called proof of work (PoW).

A few years later, in 2014, although Bitcoin already had limited programmability features, Ethereum was introduced [173] and it is, at the time of writing, among the most popular blockchain supporting *smart contracts*, *i.e.*, a software running on top of the blockchain that automatically enforces certain rules based on how parties interact with it through transactions.

In a blockchain context, and in general in a distributed system, consensus is a central concept, *i.e.*, everybody has to agree on one state (*e.g.*, the transactions history). However, the FLP impossibility [95] showed in the 1980s that consensus cannot be achieved over asynchronous networks, as in the case of a blockchain. Thus, for a blockchain we consider eventual consensus, *i.e.*, consensus is reached only asymptotically. Indeed, in 2015 a notable analysis of the Bitcoin protocol was conducted [99], introducing the concepts of *chain growth*, *i.e.*, a constant number of new blocks is added to the chain after a constant

number of rounds, *chain quality*, *i.e.*, given k blocks in a blockchain at least $c \cdot k$ blocks were generated by honest parties for a constant c with overwhelming probability and *common prefix*, *i.e.*, the probability that two honest parties see different chains after removing the k last blocks from a chain is negligible in k, and it has been formally proven under which assumptions these properties hold in the case of Bitcoin, *e.g.*, the adversary cannot control more than half of all of the computational power invested in solving the PoW puzzle. Subsequently, other provably secure blockchain protocols based on proof of stake (PoS), an energy-saving (among other advantages) alternative to PoW, have been proposed, including Ouroboros [126], Ouroboros Praos [84] and Algorand [105].

Nowadays many efficient MPC protocols for specific applications exist, however they may not prevent parties to *abort*, *i.e.*, do not properly conclude the execution of the protocol if it is not in their interest given what they learnt from the protocol execution up to a certain moment. Here is where MPC and blockchain can meet. Indeed, smart contracts can play the role of enforcing a set of predefined rules and, at last, enforcing the correct execution of an MPC protocol when parties are assumed to be rational. Thus, a mechanism design approach can be adopted to financially punish cheating parties by losing a public cryptocurrency deposit (e.g., [32, 8, 133, 132, 134, 33, 30, 83, 19, 127, 72]).

In Chapter 3, we go one step further in this direction by introducing *secret* deposits, leveraging *secret* cryptocurrency transactions and public smart contracts, and use them in the design of efficient protocols for auctions.

Then, in Chapter 4 we study the problem of front-running in the context of decentralized finance, *i.e.*, the malicious act of both manipulating the order of pending trades and injecting additional trades to make a profit at the cost of other users. Motivated by the financial loss and increased transaction load resulting from adversarial front-running, we describe common front-running attacks, propose a schema of front-running mitigation categories, assess the state-of-the-art techniques in each category and illustrate remaining attacks.

Finally, in Chapter 5 we face the issue of finding a trade-off between privacy and accountability of the users in the context of anonymous credentials. Indeed, most popular cryptocurrencies and smart contract systems offer no privacy guarantees of transactions and contract executions, while anonymous cryptocurrency systems (*e.g.*, [120]) address privacy issues, but it is impossible to investigate illegal activities conducted in the system. In particular, we propose an anonymous and generic (*i.e.*, it can be adopted outside the context of blockchain) credential scheme providing simultaneously *credentials* conditional user privacy and *auditable* privacy revocation, *i.e.*, privacy of the users can be revoked by an authority as long as it is publicly announced.

1.1 Auctions

Auctions are a common way of allocating goods or services among a set of parties based on their bids, *e.g.*, bandwidth spectrum, antiques, paintings, and slots for advertisements in the context of web search engines or social networks [75]. In

the simplest form there is a single indivisible object and each bidder has a *private valuation* for the object. One of the main desirable properties in designing an auction is *incentive compatibility*, that is the auction must be designed in a way that the participating parties can maximize their expected utilities by bidding their true valuations of the object. According to design, the auction can be categorized into open auctions and sealed-bid auctions [131].

We focus on the case of sealed-bid auctions, constructing protocols where parties holding a private bid do not have to rely on trusted third parties to ensure bid privacy. In a sealed bid auction, each bidder communicates her bid to the auctioneer privately. Then, the auctioneer is expected to declare the highest bidder as the winner and not to disclose the losing bids. In particular, in the sealed-bid *first-price* auction, the bidder submitting the highest bid wins the auction and pays what she bids, while in the sealed-bid *second-price* auction (*i.e.*, the Vickrey auction [170]) the bidder submitting the highest bid wins the auction but pays the amount of the second-highest bid [129]. It is well-known that in second-price auctions bidding truthfully is a dominant strategy, but no dominant strategy exists in the case of first-price auctions. Moreover, while in both first-price and second-price auctions a dishonest auctioneer may disclose the losing bids, the second-price auction, in particular, highly depends on trusting the auctioneer. Indeed, a dishonest auctioneer may substitute the second-highest bid with a bid that is slightly smaller than the first bid to increase her revenue. Therefore, it may not be possible or may be expensive to apply it in certain scenarios. As a result, constructing cryptographic protocols for auctioneer-free and transparent auction solutions is of great interest.

1.1.1 Related Work

Research on secure auctions started by the work of Nurmi and Salomaa [151] and Franklin and Reiter [97] in the late 1900s. However, in these first constructions, the auctioneers open all bids at the end of the protocol, which reveals the losing bids to all parties. Since then, many sealed bid auction protocols have been proposed to protect the privacy of the losing bids, *e.g.*, [5, 17, 118, 122, 139, 135]. However, in most of these protocols, privacy is obtained by distributing the computation of the final outcome to a group of auctioneers.

A lot of work has been done to remove the role of the trusted parties, including the proposed protocols by Brandt [44, 42, 43], Brandt and Sandholm [45] and Bag *et al.* [13]. In these protocols, the bidders must compute the winning bid in a joint effort through emulating the role of the auctioneer. Moreover, the seller plays a role in the auction and it is assumed that the seller has no incentive to collude with other bidding parties. However, later by Dreier *et al.* [91] it was pointed out that if the seller and a group of bidding parties collude with each other, then they can learn the bids of other parties. Besides weak security guarantees, the main drawback of the protocol proposed by Brandt [43] is that it has exponential computational and communication complexities. There has been implementations of auctions including [37, 36], which have been deployed in practice for the annual sugar beets auction in Denmark. Other works [148] have

considered the use of rational cryptography in enhancing privacy. Finally, the current state-of-the-art in protocols for secure First-Price Sealed-Bid Auctions was achieved in SEAL [13], which we compare with the protocols we propose in detail in Section 3.8. However, to the best of our knowledge, none of these works considers incentives for the parties to complete the protocol or punishment for cheaters.

An often desired feature of secure multi-party computation is that if a cheating party obtains the output, then all the honest parties should do so as well. Protocols that guarantee this are also called *fair* and are known to be impossible to achieve with dishonest majorities [74]. Recently, Andrychowicz et al. [8] (and independently Bentov & Kumaresan [32]) initiated a line of research that aims at incentivizing fairness in MPC by imposing cryptocurrency-based financial penalties on misbehaving parties. A line of work [133, 132, 134, 33, 30, 83] culminating in [19] improved the performance of this approach with respect to the amount of on-chain storage and size of the collateral deposits from each party, while others obtained stronger notions of fairness [127, 72]. However, all of these works focus on using *public* collateral deposits for incentivizing fairness, which is not possible for our application. Moreover, they rely on general-purpose MPC, while we provide a highly optimized specific purpose protocol for auctions with financial incentives. The protocols of [98, 90] are also based on cryptocurrencies. The work of [98] is the closest to ours as it leverages a cryptocurrency to ensure fairness, but it relies on SGX trusted execution enclaves which are known to be broken [169].

1.1.2 Our Contributions

In Chapter 3, we present the results published in the 20th International Conference on Applied Cryptography and Network Security (ACNS 2022) [86] and available on Cryptology ePrint Archive [85], where we propose Fair Auctions via Secret Transactions (FAST), in which there is no trusted auctioneer and where rational adversaries are always incentivized to complete protocol execution through a *secret* collateral deposit. The proposed protocol is such that each party can make sure the winning bid is the actual bid submitted by the winning party, and malicious parties can be identified, financially punished and removed from the execution (guaranteeing a winner is always determined). Our contributions are summarized as follows:

- We propose using *secret* collateral deposits dependent on private bids inputs to ensure that the optimal strategy is for parties to complete the protocol.
- (Section 3.5) We propose methods for implementing a financial punishment mechanism based on secret deposits and standard public smart contracts, which can be used to ensure the fair execution of our protocols.
- (Sections 3.6 and 3.7) We propose a cheater identifiable and publicly verifiable sealed bid auction protocols compatible with our secret deposit

approach and more efficient than the state-of-the-art [13]. Our protocols are guaranteed to terminate, finding the winner and paying the seller even if cheating occurs.

To achieve fairness in an auction setting, we require each party to provide a *secret* deposit of an amount of cryptocurrency equal to the party's private bid plus the cost of executing the protocol. In case a party is found to be cheating, a smart contract automatically redistributes cheaters' deposits among the honest parties, the cheater is eliminated and the remaining parties re-execute the protocol using their initial bids/deposits. Having a bid dependent deposit guarantees that it is always more profitable to execute the protocol honestly than to cheat (as analyzed in Section 3.9).

However, previous works that considered the use of cryptocurrency deposits for achieving fairness (*e.g.*, [32, 8, 133, 132, 134, 33, 30, 83, 19, 127, 72]) crucially rely on deposits being public, thus using the same approach would reveal information about the bid. To overcome this, we propose using secret deposits that keep the value of the deposit secret until cheating is detected. Moreover, this ensures that the parties have sufficient funds to bid for the object (*e.g.*, in a second-price auction, a party could bid very high just to figure out what the second-highest price is and then claim her submitted bid was just a mistake). Our protocols are constructed in such a way that it is possible to prove to the smart contract that a party has cheated.

We wish to emphasize that:

- While using deposits to achieve fairness represents a well-known technique, previous works considered public deposits only.
- Public deposits are not suitable for applications such as sealed-bid auctions since in order to achieve fairness, bid-dependent deposits are required, and public deposits would reveal information about the bid. For this reason, we introduce secret deposits, which represent a novel technique.
- From a sealed bid auction perspective, our protocol improves the state-of-the-art both in terms of efficiency and security guarantees, i.e., it achieves fairness (while in previous works the adversary may learn the outcome of the auction before honest parties and abort without suffering any consequences).
- No previous work in this setting considers adaptive adversaries since it would drastically increase the complexity of the protocol. For this reason, we focus on the static adversary case only.

1.2 Decentralized Finance (DeFi)

Decentralized finance (DeFi) represents an emerging alternative or complement to the current financial system. Its key feature is providing financial services,

without relying on trusted third parties such as banks or exchanges, by using smart contracts running on a blockchain.

In the context decentralized applications (Dapps) and specifically decentralized exchanges (DEXs), *i.e.*, exchanges running on a blockchain, we refer to front-running as the malicious, and often illegal, act of both manipulating the order of pending trades and injecting additional trades to make a profit at the cost of other users. In particular, front-running strategies exploit both public knowledge of user trades from transactions pending on the network and the miner's ability to determine the final transaction order.

1.2.1 Related Work

Specific instances of front-running in decentralized finance (DeFi) were first quantified by Daian et al. [78] and systematized by Eskandari et al. [93]. Besides imposing a financial penalty on honest users, front-running can also degrade the performance of blockchain networks, as recently observed on the Avalanche blockchain [10].

1.2.2 Our Contributions

In Chapter 4, we present the results that will appear in Financial Cryptography and Data Security, FC 2022 International Workshops, DeFi'22 and are available on Cryptology ePrint Archive [22]. In order to evaluate the efficacy of front-running mitigation techniques, we first formulate the set of adversarial powers which permit front-running strategies to be exploited. Concretely, if users submit their intended interaction to a pool of pending transactions, the front-running adversary has the ability to:

1. Append pending transactions to the blockchain.

2. Infer user intentions from pending transactions and blockchain state.

Then, we describe common **front-running attacks** (Section 4.1) and assess three front-running **mitigation categories** (Section 4.2) for their isolated and combined efficacy in neutralizing front-running (Figure 1.1). We introduce a speculative sandwich attack on input batching techniques (Section 4.2.2), which can be mitigated with private user balances and secret input stores (Section 4.2.3).

<table>
<tr><th>Adversarial power</th><th colspan="2">Section 4.2 Mitigation</th></tr>
<tr><td rowspan="2">1. Transaction sequencing</td><td colspan="2">Section 4.2.1 Fair ordering</td></tr>
<tr><td rowspan="2">Section 4.2.2 Batching of blinded inputs</td><td>Commit & reveal</td></tr>
<tr><td rowspan="2">2. Inference of user intent</td><td>Input aggregation</td></tr>
<tr><td colspan="2">Section 4.2.3 Private user balances & secret input store</td></tr>
</table>

Figure 1.1: Overview of mitigation techniques

Fair ordering (Section 4.2.1), implemented at the consensus protocol layer, ensures that the local receipt-order of gossiped transactions seen by a node is consistent with the final transaction ordering in the blockchain. We observe that *fair ordering* effectively mitigates the miner's ability to freely sequence transactions, but introduces a front-running adversary which rushes the network.

		User balance & input store	
		Public	**Private, secret**
Batching of blinded inputs	Commit & reveal	Speculative	Taint of user balances
	Input aggregation	Sandwich Attacks	-

Figure 1.2: Efficacy: batching of blinded inputs.

Batching of blinded inputs (Section 4.2.2) replaces the *sequential* model of DeFi interaction with a *round-based* one, where user inputs are blinded in each round to ensure input independence, thereby thwarting front-running strategies that rely on prior knowledge of other users' intentions. However, if user balances are public, the input may still be partially inferred when the valid user's input space is constrained by its balance: here, we contribute a novel, *speculative* front-running attack that exploits the *direction* of an automated market maker (AMM) swap, leaked from the victim's public balance. Furthermore, we highlight differences between *commit & reveal* and *input aggregation* approaches to batching of blinded inputs (Figure 1.2). In commit & reveal schemes, user inputs are revealed *individually*: Although front-running in the specific round is no longer possible, they necessarily leak information about the subsequent balance-update for each participating user, even if the user balances are private. If the taint of private balances is sufficiently strong, this can allow the front-running adversary to infer the users future inputs (*e.g.*, the intended AMM swap direction).

Private user balances (Section 4.2.3) are thus necessary to prevent the leakage of the valid user input space from balances and application state. Although DeFi state must generally remain public to retain its utility [9], we show that it is necessary to shield certain fragments thereof which explicitly reveal future user intent. **Secret input stores** (Section 4.2.3) protect inputs that are evaluated by the application after a time delay [172] or, in the case of order books, whenever a match with other user inputs [77, 21] can be found.

1.3 Anonymous Credentials

Ensuring user privacy while complying with requirements for user accountability is often a challenging task. As an example, consider an on-line payment platform. User privacy demands that identities remain unknown while performing on-line payments, while Know Your Customer and Anti-money laundering regulations demand that misbehaving users should be held accountable.

This and many more sophisticated examples motivate the analysis of the trade-offs between user privacy and accountability, both from a technical perspective [54, 55, 96, 110, 165], and from an ethical standpoint [1, 119, 168].

The notion of conditional privacy captures settings where a set of authorities is given the power to revoke a user's privacy. Unfortunately, the vast majority of existing systems that provide conditional privacy naïvely trust revocation authorities to trigger privacy revocation only in case a user behaves suspiciously. Thus, they do not hold authorities accountable, allowing them to surreptitiously revoke privacy. In particular, third party auditors (*e.g.*, regulatory agencies and users themselves) cannot verify whether privacy revocation has happened (or not). As a consequence, user trust in the privacy of such systems is eroded.

We address this issue by introducing the notion of Publicly Auditable Privacy Revocation (PAPR). In schemes offering conditional privacy, PAPR makes the actions of authorities transparent to third party auditors, who can monitor when privacy revocation takes place and thus detect abuse of power by the authorities. We showcase the power (and challengers) of this notion by showing how to add PAPR to anonymous credential schemes in order to achieve increased (user) trust via strong accountability guarantees for both users and authorities.

1.3.1 Related Works

Privacy Preserving Authentication allows users to authenticate without revealing their true identities. This feature is crucial for systems with strong user privacy requirements, and can be achieved in many ways. Anonymous credentials, envisioned by Chaum in [66] and first realized with provably security in [56], allow a user to prove ownership of a valid credential without revealing their identity. Later, anonymous credential schemes with improved efficiency [57, 25, 14] were proposed. Schemes with richer features such delegation [76] and attributes [57, 14, 35] have also been proposed. More recently, universally composable [60] anonymous credentials were proposed in [52, 51]. In anonymous credential schemes, there are two main strategies to prevent abuse of anonymity: allow users to authenticate anonymously only a predetermined number of times [167, 53]; or introduce mechanisms for privacy revocation by a central authority [56].

Conditional Privacy (or *revocable privacy* [165]) combines user anonymity and accountability, so that it is possible for an authority to revoke a user's right to privacy, should the target user behave in illicit ways. This is often implemented by giving a selected group of trusted entities the power to revoke confidentiality or anonymity guarantees as needed. In order to avoid malicious strategies, there is an unwillingness by authorities to let users decide who these trusted parties should be. Instead, a set of central *privacy revocation authorities* is often used. This is the case in many applications, including encryption systems [147], e-cash [41], blind signatures [166] and group signatures [68].

Publicly Auditable Privacy Revocation was introduced as a way to make authorities accountable for the act of privacy revocation and thereby prevent abuse of power. Techniques for auditing privacy revocation are often application specific. Examples include auditing the behaviour of pseudonym conversion authorities [55] or auditing that certificate authorities provide correct public keys [146]. Known approaches to obtain auditability for revocation authorities in the context of anonymous credentials either use non-standard techniques, such as witness encryption [112], or rely on a set of trusted authorities that are assumed not to collude [41, 68, 141, 147, 166].

Concurrent Works which addresses a similar goal of authority accountability was proposed in [88]. However, this scheme does not achieve any notion of composability and cannot be easily proven UC secure. Moreover, the committee that is expected to cooperate in order to revoke privacy is not hidden, so its publicly known members may be corrupted by a proactive adversary.

Anonymous Committees address the problem of ensuring that a set of parties do not collude, by establishing a committee whose members' identities are not known to any party, including the committee members themselves (*i.e.*, a member knows it is in the committee but does not know the identity of other members). Several works exist on this problem, *e.g.*, [69, 126, 84, 79]. In this setting, it is both hard for committee members to collude and for an adversary to subvert committee members. In particular, the idea of distributing sensitive information to anonymous committees(*e.g.*, privacy revocation trapdoors) or having anonymous committees execute cryptographic protocols has been explored in the context of proactive secret sharing [31, 111, 65], multiparty computation (MPC) [103] and threshold encryption [92]. These protocols work in the so called You Only Speak Once (YOSO) model, where a fresh randomly chosen anonymous committee executes each round of the protocol, limiting the adversary to probabilistic corruptions (*i.e.*, when the adversary corrupts any party, it only knows that this party may be party of the current committee with a certain probability smaller than 1).

1.3.2 Our Contributions

In Chapter 5, we present the results that will appear in CT-RSA 2023, Cryptographers' Track at RSA Conference [47], where we introduce the concept of anonymous credentials with PAPR, which we model and construct in the Universal Composability [60] framework. We define this new concept as an ideal functionality supporting standard actions of anonymous credentials issuance, linkable[1] credential showing and privacy revocation. Our ideal functionality captures the novel PAPR property by guaranteeing that all parties are notified

[1]While many anonymous credential schemes strive to provide unlinkability among different showings, we restrict ourselves to the simpler case where different showings of the same credential can be linked in order to focus on our new PAPR techniques.

when the issuer performs privacy revocation on a credential. Enforcing this guarantee is the main challenge in our construction.

The core of our contribution is a novel mechanism to distributively store the secret identity connected to a user's anonymous credential in such a way that privacy revocation is possible, but any attempt to revoke privacy (by retrieving the user's identity) requires a public announcement of the privacy revocation act of the corresponding credential. Our contributions are summarized as follows:

- We introduce the notion of Publicly Auditable Privacy Revocation (PAPR) for anonymous credential schemes.
- We provide a security definition of anonymous credentials with PAPR in the Universal Composability framework (Section 5.2).
- We construct an efficient anonymous credential scheme that achieves our PAPR notion with UC security against static malicious adversaries under standard assumptions (Section 5.3).
- We show how to modify our construction to obtain a PAPR anonymous credential scheme that is UC-secure against mobile adversaries via proactive secret sharing and threshold encryption in the YOSO model (Section 5.4).

Chapter 2

Preliminaries

Let $y \xleftarrow{\$} F(x)$ denote running the randomized algorithm F with input x and implicit randomness, obtaining the output y. When the randomness r is specified, we use $y \leftarrow F(x;r)$. For a set $\mathcal{X}$, let $x \xleftarrow{\$} \mathcal{X}$ denote x chosen uniformly at random from $\mathcal{X}$; and for a distribution $\mathcal{Y}$, let $y \xleftarrow{\$} \mathcal{Y}$ denote y sampled according to the distribution $\mathcal{Y}$. We denote concatenation of two values x and y by $x|y$. A function $f(x)$ is negligible in x if $f(x)$ is positive and for every positive polynomial $p(x) \in \text{poly}(x)$ there exists a $x' \in \mathbb{N}$ such that $\forall x \geq x' : \ f(x) < 1/p(x)$, we denote such functions as $\text{negl}(x)$. Two ensembles $X = \{X_{\lambda,z}\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$ and $Y = \{Y_{\lambda,z}\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$ of binary random variables are said to be *computationally indistinguishable*, denoted by $X \approx_c Y$, if for all z it holds that $\mid \Pr[\mathcal{D}(X_{\lambda,z}) = 1] - \Pr[\mathcal{D}(Y_{\lambda,z}) = 1] \mid$ is negligible in the security parameter λ for every non-uniform probabilistic polynomial-time (PPT) distinguisher $\mathcal{D}$. For a field $\mathbb{F}$ we denote by $\mathbb{F}[X]_{\leq m}$ the vector space of polynomials in $\mathbb{F}[X]$ of degree at most m. Let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^l$ denote a collision resistant hash function that maps the binary representation of the argument to a binary string of length l. Let $\lambda \in \mathbb{N}$ denote a security parameter. Finally, we use the notation $\vec{a}[i]$ to denote the i'th element of the vector $\vec{a}$. When signing messages not in the message space of the signature algorithm (*e.g.*, a group element or a vector), we let the conversion to the message space be implicit.

2.1 Models

2.1.1 Adversarial Models

In our work we consider different adversarial models:

- *Passive adversary*: does not deviate from the protocol but executes any polynomial time computation on the messages and the internal state of the corrupted parties in order to learn private information.
- *Active adversary*: deviates from the protocol in arbitrary ways.

- *Rational adversary*: deviates from the protocol only if profitable.

Moreover, we mainly consider the *static* restriction, *i.e.*, the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. We also consider the *mobile* case, that is defined in Chapter 5.

2.1.2 Random Oracle Model (ROM)

A *random oracle* [27] is a map $R : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ such that for each unique input returns an output chosen uniformly at random (with λ a security parameter). In case the random oracle is queried more than once with the same input, it returns the same output. A protocol using this abstraction is said to use the *random oracle model*.

2.2 Frameworks

2.2.1 Real/Ideal Simulation Paradigm with Sequential Composition

This paradigm is commonly used to analyse cryptographic protocol security and provides strong security guarantees, namely that several instance of the protocol can be executed in sequence while preserving their security. In order to prove security, a real world and an ideal world are defined and compared. In the real world, the protocol π is executed with the parties, some of which are corrupted and controlled by the adversary $\mathcal{A}$. In the ideal world the protocol is replaced by an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ interacts with it. The ideal functionality $\mathcal{F}$ describes the behaviour that is expected from the protocol and acts as a trusted entity. A protocol π is said to securely realize the ideal functionality $\mathcal{F}$, if for every polynomial-time adversary $\mathcal{A}$ in the real world, there is a polynomial-time simulator $\mathcal{S}$ for the ideal world, such that the two worlds cannot be distinguished. In more detail, no probabilistic polynomial-time distinguisher $\mathcal{D}$ can have a non-negligible advantage in distinguishing the concatenation of the output of the honest parties and of the adversary $\mathcal{A}$ in the real world from the concatenation of the output of the honest parties (which come directly from $\mathcal{F}$) and of the simulator $\mathcal{S}$ in the ideal world. More details about this model can be found in [59, 106].

2.2.2 Universally Composable Security

In the Universal Composability (UC) framework [60] the security of a protocol is analyzed under the real-world/ideal-world paradigm, *i.e.*, by comparing the real world execution of a protocol with an ideal world interaction with the ideal functionality that it realizes. Protocols that are secure in the UC framework can be arbitrarily composed with each other without compromising security. In the ideal world execution, dummy parties (potentially controlled by an ideal

adversary $\mathcal{S}$, referred to as the simulator) interact with an ideal functionality $\mathcal{F}$. In the real world execution, parties (potentially corrupted by a real world adversary $\mathcal{A}$) interact with each other by following a protocol π that realizes the ideal functionality $\mathcal{F}$. The real and ideal executions are controlled by the environment $\mathcal{Z}$, an entity that controls inputs and reads the outputs of each party, $\mathcal{A}$ and $\mathcal{S}$. The protocol π securely realizes $\mathcal{F}$ in the UC framework if the environment $\mathcal{Z}$ cannot efficiently distinguish between the real world execution with π and $\mathcal{A}$ and the ideal world execution with $\mathcal{S}$ and $\mathcal{F}$.

2.3 Assumptions

2.3.1 Decisional Diffie Hellman (DDH) Assumption

The DDH problem consists in deciding whether $c = ab$ or $c \xleftarrow{\$} \mathbb{Z}_p$ in a tuple (g, g^a, g^b, g^c) where g is a generator of a group $\mathbb{G}$ of order p, and $a, b \xleftarrow{\$} \mathbb{Z}_p$. The DDH assumption states that the DDH problem is hard for every *PPT* distinguisher. It is well known that the DDH assumption implies the Discrete Logarithm assumption.

2.3.2 Discrete Logarithm (DL) Assumption

Let $\mathbb{G}$ be a finite cyclic group of prime order p and let $g \in \mathbb{G}$ be a generator. Given $h \in \mathbb{G}$, finding $x \in \mathbb{Z}_q$ such that $g^x = h$ is computationally hard for any *PPT* algorithm.

2.4 Cryptographic primitives

2.4.1 Commitment Scheme

Definition 1 (Commitment Scheme)**.** *A commitment scheme* C *is a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ *defined as follows.*

$\mathsf{C.Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp *(implicit input in all subsequent algorithms)*

$\mathsf{C.Commit}(m, r) \to c$*: This procedure takes as input a vector of messages* $m = \{m_1, \ldots, m_n\}$ *and a vector of randomness* $r = \{r_1, \ldots, r_n\}$ *(sometimes referred to as key). It outputs a commitment* c*.*

$\mathsf{C.Open}(c, m, r) \to v$*: This procedure takes as input a commitment* c*, a vector of messages* $m = \{m_1, \ldots, m_n\}$ *and a vector of randomness* $r = \{r_1, \ldots, r_n\}$*. It outputs a verification bit indicating whether* m, r *is an opening to* c *or not.*

A commitment scheme should be *correct*, i.e the opening procedure will return 1 (accept) with probability 1 if the commitment c is generated by Commit on the remainder of the input to Open. Furthermore, it should be *hiding*, in the sense that the commitment leaks no information about the message, and *binding* so that the commitment can only be opened to the committed message. These properties are defined in [4, Def. 1, Def. 2].

Pedersen commitments

Let p and q be large primes such that q divides $p-1$ and let $\mathbb{G}$ be the unique subgroup of $\mathbb{Z}_p^*$ of order q. All the computations in $\mathbb{G}$ are operations modulo p, however we omit the mod p to simplify the notation. Let g, h denote random generators of $\mathbb{G}$ such that nobody knows the discrete logarithm of h base g, *i.e.*, a value w such that $g^w = h$. The Pedersen commitment scheme [156] to an $s \in \mathbb{Z}_q$ is obtained by sampling $t \xleftarrow{\$} \mathbb{Z}_q$ and computing $\mathsf{com}(s,t) = g^s h^t$. Hence, the commitment $\mathsf{com}(s,t)$ is a value uniformly distributed in $\mathbb{G}$ and opening the commitment requires to reveal the values of s and t. The Pedersen commitments are additively homomorphic, *i.e.*, starting from the commitment to $s_1 \in \mathbb{Z}_q$ and $s_2 \in \mathbb{Z}_q$, it is possible to compute a commitment to $s_1 + s_2 \in \mathbb{Z}_q$, *i.e.*, $\mathsf{com}(s_1,t_1) \cdot \mathsf{com}(s_2,t_2) = \mathsf{com}(s_1+s_2, t_1+t_2)$.

2.4.2 Public Key Encryption Scheme

Definition 2 (Public Key Encryption Scheme)**.** *A public-key encryption scheme is a tuple of PPT algorithms* (*Setup*, *KeyGen*, *Encrypt*, *Decrypt*) *defined as follows.*

Enc.Setup(1^λ)*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp*.*

Enc.KeyGen$(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$*: The key generation algorithm takes as input the public parameters and outputs a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

Enc.Encrypt$(m, \mathsf{pk}) \to c$*: The encryption algorithm takes as input a message* m *and a public key* pk*. It returns a ciphertext* c*.*

Enc.Decrypt$(c, \mathsf{sk}) \to m$*: The decryption algorithm takes as input a ciphertext* c *and a secret key* sk*. It returns a plaintext message* m*.*

We consider an encryption scheme secure if it is indistinguishable under Chosen Plaintext Attacks (IND-CPA) and key-private as formalised in [26, Def. 1].

2.4.3 Digital Signatures

Definition 3 (Signature Scheme)**.** *A signature scheme Sig with message space* $\mathcal{M}$ *is a tuple of PPT algorithms* (*Setup*, *KeyGen*, *Sign*, *Verify*) *defined as follows.*

Sig.Setup(1^λ)*: The setup algorithm takes as input the security parameter and outputs some public parameters* pp*.*

Sig.KeyGen$(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$*: The key generation algorithm takes as input the public parameters (or the security parameter* λ *only) and outputs a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*.*

Sig.Sign$(\mathsf{sk}, m) = \mathit{Sig.Sign}_{\mathsf{sk}}(m) \to \sigma$*: The sign algorithm takes in input a secret key* sk *and a message* $m \in \mathcal{M}$*; it outputs a signature* σ*.*

Sig.Verify$(\mathsf{pk}, m, \sigma) \to v$*: The verification algorithm takes in input a public key* pk*, a message* $m \in \mathcal{M}$ *and signature* σ*. It outputs 0 (reject) or 1 (accept).*

Throughout the thesis we assume Sig to be *correct* and *existentially unforgeable* as in [38, Def. 2]. In a nutshell this means that signatures generated by

the signing algorithm are always accepted by the verifying algorithm, and that without the knowledge of the secret key it is computationally infeasible to generate a signature that is accepted by the verifying algorithm with non-negligible probability.

2.4.4 Non-interactive Zero-Knowledge Proofs of Knowledge

Let $\mathcal{L}$ be a language in NP and $R_{\mathcal{L}}$ be a relationship such that $\mathcal{L} = \{x \mid \exists w : (x, w) \in R_{\mathcal{L}}\}$. A zero-knowledge proof of knowledge protocol $\Sigma = (P, V)$ for a language $\mathcal{L}$ allows a prover P to demonstrate to a verifier V that $x \in \mathcal{L}$ provided that the prover knows a witness w such that $(x, w) \in R_{\mathcal{L}}$ and with the following properties:

(Completeness) If $(x, w) \in R_{\mathcal{L}}$ then the proof generated by P is accepted by the verifier V with overwhelming probability;

(Soundness) It is computationally hard for the prover P to prove to the verifier V that $(x, w) \in R_{\mathcal{L}}$ when $(x, w) \notin R_{\mathcal{L}}$;

(Zero knowledge) There exists a PPT simulator S that, by using rewinding, takes as input x but not w and generates a proof that $(x, .) \in R_{\mathcal{L}}$ for some w, *i.e.*, the verifier V does not learn w but can check if $(x, .) \in R_{\mathcal{L}}$ for some w;

(Proof of knowledge) There exists a PPT simulator S that interacts with a copy of the prover P and extracts, by using rewinding, the witness w, *i.e.*, the verifier V can check if the prover P knows a witness w such that $(x, w) \in R_{\mathcal{L}}$;

2.4.5 Publicly Verifiable Secret Sharing (PVSS)

In our work, we use the PVSS protocol π_{PVSS} from [64], which is described in detail in Figure 2.1. A PVSS protocol allows for a dealer to distribute encrypted shares to a set of parties in such a way that only one specific party can decrypt a share but any third party verifier can check that all shares are valid. Later on, each party can decrypt its corresponding share to allow for reconstruction while showing to any third party verifier that the decrypted share corresponds to one of the initial encrypted shares. A deposit committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ will execute this protocol verifying and decrypting shares provided as part of our secret deposit mechanism (further discussed in Section 3.5). Since the parties in $\mathcal{C}$ executing π_{PVSS} must have public keys registered as part of a setup phase, we capture this requirement in $\mathcal{F}_{\mathsf{SC}}$ as presented in Section 3.4.

In order to instantiate π_{PVSS}, the NIZKs described below are also necessary:

- **NIZK for Discrete Logarithm Equality (DLEQ):** This NIZK from [67] is used to prove that, given $g_1, ..., g_m$ and $x_1, ..., x_m$, the discrete logarithms of every x_i with base g_i are equal, *i.e.*, $x_i = g_i^{\alpha}$ for all $i = 1, \ldots, m$ for some $\alpha \in \mathbb{Z}_q$ (the same α for all i). It is denoted as $DLEQ((g_i, x_i)_I, (h_i, y_i)_i)$. In this NIZK, the prover computes $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$,

for $H(\cdot)$ a random oracle (that will be instantiated by a cryptographic hash function) and z as above. The proof is $(a_1, \ldots, a_m, e, z)$. The verifier checks that $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$ and that $a_i = g_i^z x_i^e$ for all i.

- **NIZK for Low-Degree Exponent Interpolation ($LDEI$):** This NIZK from [64] is used to prove that, given generators $g_1, g_2, \ldots, g_m$ of a cyclic group $\mathbb{G}_q$ of prime order q, pairwise distinct elements $\alpha_1, \alpha_2, \ldots, \alpha_m$ in $\mathbb{Z}_q$ and an integer $1 \leq k < m$ known by a prover and a verifier, for $p(x)$ known by the prover, it holds that $(x_1, x_2, \ldots, x_m) \in \{(g_1^{p(\alpha_1)}, g_2^{p(\alpha_2)}, \ldots, g_m^{p(\alpha_m)}) : p \in \mathbb{Z}_q[X], \deg p \leq k\}$. We denote this NIZK by $LDEI((g_i)_{i\in[m]}, (\alpha_i)_{i\in[m]}, k, (x_i)_{i\in[m]})$. In this NIZK, the sender chooses $r \in \mathbb{Z}_q[X]_{\leq k}$ uniformly at random and computes $a_i = g_i^{r(\alpha_i)}$ for all $i = 1, \ldots, m$, $e = H(x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)$ and $z = e \cdot p + r$. The proof is then $(a_1, a_2, \ldots, a_m, e, z)$. The verifier checks that $z \in \mathbb{Z}_q[X]_{\leq k}$, that $x_i^e \cdot a_i = g_i^{z(\alpha_i)}$ holds for all $i = 1, \ldots, m$ and that $e = H(x_1, x_2, \ldots, x_m, a_1, a_2, \ldots, a_m)$.

Definition 4 (Definition 4 from [64]). ***Indistinguishability of secrets (IND1-secrecy)*** *We say that the PVSS is IND1-secret if for any polynomial-time adversary $\mathcal{A}$ corrupting at most $t-1$ parties, $\mathcal{A}$ has a negligible advantage in the following game played against a challenger.*

1. *The challenger runs the Setup phase of the PVSS as the dealer and sends all public information to $\mathcal{A}$. Moreover, it creates secret and public keys for all honest parties and sends the corresponding public keys to $\mathcal{A}$.*
2. *$\mathcal{A}$ creates secret keys for the corrupted parties and sends the corresponding public keys to the challenger.*
3. *The challenger chooses values $\mathbf{x}_0$ and $\mathbf{x}_1$ at random in the space of secrets. Furthermore it chooses $b \leftarrow \{0, 1\}$ uniformly at random. It runs the Distribution phase of the protocol with $\mathbf{x}_0$ as secret. It sends $\mathcal{A}$ all public information generated in that phase, together with $\mathbf{x}_b$.*
4. *$\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$.*

The advantage of $\mathcal{A}$ is defined as $|\Pr[b = b'] - 1/2|$.

Proposition 1 (Proposition 3 from [64]). *Protocol π_{PVSS} is IND1-secret under the DDH assumption.*

2.4.6 Blind Signature

Definition 5 (Blind signature). *A blind signature BSig with message space $\mathcal{M}$ is a tuple of PPT algorithms (KeyGen, User, Sign, Verify) defined as follows ([2]).*

$\mathsf{BSig.KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: The randomized key generation algorithm takes as input a security parameter 1^λ with $\lambda \in \mathbb{N}$ and outputs a secret/public key pair $(\mathsf{sk}, \mathsf{pk})$.

Protocol π_{PVSS} from [64]

Let h be a generator of a group $\mathbb{G}_q$ of order q. Let $H(\cdot)$ be a random oracle. Protocol π_{PVSS} is run between n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, a dealer D and an external verifier V (in fact any number of external verifiers) who have access to a public ledger where they can post information for later verification.

1. **Setup:** Party $\mathcal{P}_i$ generates a secret key $\mathsf{sk}_i \leftarrow \mathbb{Z}_q$, a public key $\mathsf{pk}_i = h^{\mathsf{sk}_i}$ and registers the public key pk_i by posting it to the public ledger, for $1 \leq i \leq n$.
2. **Distribution:** The dealer D samples a polynomial $p(X) \leftarrow \mathbb{Z}_q[X]_{\leq t+\ell-1}$ (where $t = \lceil \frac{n}{2} \rceil - 1$ and ℓ is the number of secrets) and sets $s_0 = p(0), s_1 = p(-1), \ldots, s_{\ell-1} = p(-(\ell-1))$. The secrets are defined to be $S_0 = h^{s_0}, S_1 = h^{s_1}, \ldots, S_{\ell-1} = h^{s_{\ell-1}}$. D computes Shamir shares $\sigma_i = p(i)$ for $1 \leq i \leq n$. D encrypts the shares as $\hat{\sigma}_i = \mathsf{pk}_i^{\sigma_i}$ and publishes $(\hat{\sigma}_1, \ldots, \hat{\sigma}_n)$ in the public ledger along with the proof $LDEI$ that $\hat{\sigma}_i = \mathsf{pk}_i^{p(i)}$ for some p of degree at most $t+\ell-1$.
3. **Verification:** The verifier checks the proof $LDEI$.
4. **Share decryption**[a] **(for $\mathcal{P}_i$):** On input $\hat{\sigma}_i, \mathsf{pk}_i$, decrypt share $\tilde{\sigma}_i = \hat{\sigma}_i^{\frac{1}{\mathsf{sk}_i}} = h^{\sigma_i}$ and publish it in the ledger together with $PROOF_i = DLEQ((h, \tilde{\sigma}_i), (\mathsf{pk}_i, \hat{\sigma}_i))$ (showing that the decrypted share $\tilde{\sigma}_i$ corresponds to $\hat{\sigma}_i$).
5. **Share decryption verification:** Apply the verification algorithm of the DLEQ proof $PROOF_i$ and complain if this is not correct.
6. **Secret reconstruction algorithm $Rec_{\mathcal{Q}}$:** On input $\{\tilde{\sigma}_i\}_{i \in \mathcal{Q}}$ for a set $\mathcal{Q}$ of exactly $n-t$ indices, for $j \in [\ell-1]$, set $\lambda_i^{(j)} = \prod_{m:m \in \mathcal{Q}, m \neq i} \frac{-j-m}{i-m}$ for all $i \in \mathcal{Q}$ and compute $S_j = \prod_{i \in \mathcal{Q}} (\tilde{\sigma}_i)^{\lambda_i^{(j)}} = \prod_{i \in \mathcal{Q}} h^{p(i)\lambda_i^{(j)}} = h^{p(-j)} = h^{s_j}$, and publish the values S_j.

[a] Amortized share decryption (for $\mathcal{P}_i$): If the PVSS has been used several times where $\mathcal{P}_i$ has received in each case a share $\hat{\sigma}_i^a$, $\mathcal{P}_i$ can decrypt shares as above but publish one single proof $PROOF_i = DLEQ((h, (\tilde{\sigma}_i^a)_a), (\mathsf{pk}_i, (\hat{\sigma}_i^a)_a))$.

Figure 2.1: Protocol π_{PVSS} from [64]

BSig.User and BSig.Sign are randomized interactive algorithms.

The user runs BSig.User on an initial state (pk, m)*, where* pk *is a public key and* $m \in \mathcal{M}$ *is a message, and let it interact with BSig.Sign run by the signer on initial state a secret key* (sk)*. At the end of the protocol, BSig.User either enters the* `halt` *state and outputs a signature* σ *as its last outgoing message, or enters the* `fail` *state. Instead, BSig.Sign simply enters the* `halt` *state, without generating any output.*

BSig.Verify$(\mathsf{pk}, m, \sigma) \rightarrow v$*: The deterministic verification algorithm takes in input a public key* pk*, a message* $m \in \mathcal{M}$ *and signature* σ*. It outputs 0 (reject) or 1 (accept).*

Where an *interactive algorithm* is a stateful algorithm that on input an incoming message m_{in} (this is ϵ if the party is initiating the protocol) and state St outputs an outgoing message m_{out} and updated state St'. Two interactive algorithms `A` and `B` are said to *interact* when the outgoing messages of `A` are passed as incoming messages to `B`, and vice versa, until both algorithms enter

either the `halt` or `fail` state. We write $(m_{\mathtt{A}}, St_{\mathtt{A}}, m_{\mathtt{B}}, St_{\mathtt{B}}) \leftarrow [\mathtt{A}(St_{\mathtt{A}}) \leftrightarrow \mathtt{B}(St_{\mathtt{A}})]$ to denote the final state after an interaction between `A` and `B` when run on initial states $St_{\mathtt{A}}$ and $St_{\mathtt{B}}$, respectively.

(Correcteness) A blind signature scheme is *correct* if for all $\lambda \in \mathbb{N}$ and for all $m \in \mathcal{M}$, it holds that $St_{\mathsf{BSig.User}} = \mathtt{halt}$ and $\mathsf{BSig.Verify}(\mathsf{pk}, m, \sigma) \rightarrow 1$ when $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and $(m_{\mathsf{Sign}}, St_{\mathsf{Sign}}, \sigma, St_{\mathsf{User}}) \leftarrow [\mathsf{BSig.Sign}(\mathsf{sk}) \leftrightarrow \mathsf{BSig.User}((\mathsf{pk}, m))]$ with probability 1.

A blind signature has to guarantee *unforgeability*, *i.e.*, a user should not be able to forge signatures, and *blindness*, *i.e.*, the signer should not be able to see the messages that is being signed, or even be able to relate signed messages to previous protocol sessions.

(Unforgeability) Let $\mathsf{BSig} = (\mathsf{KeyGen}, \mathsf{User}, \mathsf{Sign}, \mathsf{Verify})$ be a blind signature scheme, let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathcal{A}$ be a forging algorithm with access to the signing oracle. The experiment $\mathbf{Exp}^{\mathrm{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ first generates a keypair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and runs $\mathcal{A}$ on input $(1^\lambda, \mathsf{pk})$. The adversary has access to a signing oracle that runs the $\mathsf{BSig.Sign}(\mathsf{sk}, .)$ algorithm and maintains state across invocations. At the end of its execution, the adversary outputs a set of message signatures pairs $\{(m_1, \sigma_1), \ldots, (m_s, \sigma_s)\}$. Let t be the number of completed signing sessions during $\mathcal{A}$'s attack. Then $\mathcal{A}$ is said to win the game if $\mathsf{BSig.Verify}(\mathsf{pk}, m_i, \sigma_i) \rightarrow 1$ for all $1 \leq i \leq s$, all m_i are different and $s > t$. BSig is said to be *unforgeable* (*one-more unforgeable under sequential attacks*) if the probability of winning the above game is negligible for all PPT adversaries $\mathcal{A}$.

Formally, the experiment is defined as follows:

Experiment $\mathbf{Exp}^{\mathrm{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$:
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$
$SSet \leftarrow \emptyset$; $s \leftarrow 0$ // set of signer sessions and number of finished sessions
$\{(m_1, \sigma_1), \ldots, (m_s, \sigma_s)\} \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk} : \mathsf{Sign}(., .))$
if $\mathsf{BSig.Verify}(\mathsf{pk}, m_i, \sigma_i) \rightarrow 1$ for all $1 \leq i \leq s$ and $s \geq$ t and $m_i \neq m_j$ for all $1 \leq i < j \leq s$ **then**
  Return 1
else
  Return 0
end if

Where $\mathcal{A}$'s queries to the signing oracles are answered as follows:

Oracle $\mathsf{Sign}(s, m_{\mathrm{in}})$: // s is a session identifier
if $s \notin SSet$ **then**
  $SSet \leftarrow \{s\}$; $St_{\mathsf{BSig.Sign}}[s] \leftarrow \mathsf{sk}$
  $(m_{\mathrm{out}}, St_{\mathsf{BSig.Sign}}[s]) \leftarrow \mathsf{BSig.Sign}(m_{\mathrm{in}}, St_{\mathsf{BSig.Sign}}[s])$
  if $St_{\mathsf{BSig.Sign}}[s] = \mathtt{halt}$ **then**
    $t \leftarrow t + 1$

end if
Return m_{out}
end if

The advantage of $\mathcal{A}$ in breaking BSig is defined as the probability that the above experiment returns 1:

$$\mathbf{Adv}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = Pr[\mathbf{Exp}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 1]$$

BSig is said to be *one-more unforgeable under sequential attacks* if the advantage $\mathbf{Adv}^{\text{omu}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ is a negligible function in the security parameter λ for all PPT adversaries $\mathcal{A}$.

(Blindness) Let $\mathsf{BSig} = (\mathsf{KeyGen}, \mathsf{User}, \mathsf{Sign}, \mathsf{Verify})$ be a blind signature scheme, let $\lambda \in \mathbb{N}$ be the security parameter and let $\mathcal{A}$ an adversary. The adversary $\mathcal{A}$ acts as a cheating signer, who is trying to distinguish between two signatures created in different signing sessions. The experiment chooses a random bit b, generates a fresh key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ and runs the adversary $\mathcal{A}$ on input $(1^\lambda, \mathsf{pk}, \mathsf{sk})$. $\mathcal{A}$ outputs two challenge messages m_0 and m_1 and then act as the signer in two sequential interactions with the $\mathsf{BSig.User}$ algorithm. If $b = 0$, then $\mathcal{A}$ first interacts with $\mathsf{BSig.User}(\mathsf{pk}, m_0)$ and then with $\mathsf{BSig.User}(\mathsf{pk}, m_1)$; If $b = 1$, then $\mathcal{A}$ first interacts with $\mathsf{BSig.User}(\mathsf{pk}, m_1)$ and then with $\mathsf{BSig.User}(\mathsf{pk}, m_0)$. If in both sessions the $\mathsf{BSig.User}$ algorithms accept, then $\mathcal{A}$ is additionally given the resulting signatures σ_0, σ_1 for messages m_0, m_1. Finally, $\mathcal{A}$ outputs its guess d and wins the game if $b = d$. BSig is said to be *blind* (*blind under sequential attacks*) if $2p - 1$, where p is the probability that $\mathcal{A}$ wins the above game, is negligible for all PPT adversaries $\mathcal{A}$.

Formally, the experiment is defined as follows:

Experiment $\mathbf{Exp}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$:
$b \leftarrow \{0,1\}$; $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$
$((m_0, m_1), St_{\mathcal{A}}) \leftarrow \mathcal{A}(\epsilon, (1^\lambda, \mathsf{pk}, \mathsf{sk}))$
$(m_{\mathcal{A}}, St_{\mathcal{A}}, \sigma_b, St_b) \leftarrow [\mathcal{A}(St_{\mathcal{A}}) \leftrightarrow \mathsf{User}((\mathsf{pk}, m_b))]$
$(m_{\mathcal{A}}, St_{\mathcal{A}}, \sigma_{1-b}, St_{1-b}) \leftarrow [\mathcal{A}(St_{\mathcal{A}}) \leftrightarrow \mathsf{User}((\mathsf{pk}, m_{1-b}))]$
if $St_0 =$ `fail` or $St_1 =$ `fail` **then**
  $\sigma \leftarrow$ `fail`
else
  $\sigma \leftarrow (\sigma_0, \sigma_1)$
end if
$d \leftarrow \mathcal{A}(\sigma, St_{\mathcal{A}})$
if b = d **then**
  Return 1
else
  Return 0
end if

The advantage of $\mathcal{A}$ in breaking BSig is defined as

$$\mathbf{Adv}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 2 \cdot \Pr[\mathbf{Exp}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda) = 1] - 1$$

BSig is said to be *blind under sequential attacks* if the advantage $\mathbf{Adv}^{\text{blind}}_{\mathsf{BSig},\mathcal{A}}(\lambda)$ is a negligible function in the security parameter λ for all PPT adversaries $\mathcal{A}$.

2.4.7 Provable Shuffle of Commitments

Definition 6 (Provable Shuffle of Commitments)**.** *A proof system* $\mathsf{Shuf} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for proving shuffle of commitments generated by a commitment scheme* C *consists of the following algorithms.*

$\mathsf{Shuf.Setup}(1^\lambda)$*: The setup algorithm takes as input the security parameter and outputs public parameters* pp*, often referred to as the common reference string (implicitly input to all subsequent algorithms).*

$\mathsf{Shuf.Prove}(n, \rho, \{c_i\}_{i\in[n]}) \rightarrow (\{c'_i\}_{i\in[n]}, \pi)$*: The provable shuffle algorithm takes as input an integer* n*, a permutation* ρ *over the set* $\{1, \ldots, n\}$*, and* n *commitments* $\{c_i\}_{i\in[n]}$ *generated by* $\mathsf{C.Commit}$*. It returns a list of* n *commitments* $\{c'_i\}_{i\in[n]}$ *and a proof* π*.*

$\mathsf{Shuf.Verify}(n, \{c_i\}_{i\in[n]}, \{c'_i\}_{i\in[n]}, \pi) \rightarrow v$*: The verification algorithm takes as input an integer* n*, two sets of* n *commitments and a proof* π*. It returns 1 (accept) if* π *is a valid proof for the relation "there exists a set* $M = \{m_i\}_{i\in[n]}$ *and a permutation* $\rho \in S_n$ *s.t.* $\{\mathsf{C.Open}(c_i, m_i, r_i)\}_{i\in[n]} = \{\mathsf{C.Open}(c'_{\rho(i)}, m_{\rho(i)}, r'_{\rho_i}\}_{i\in[n]}$*", where the randomnesses* r_i, r'_i *are extracted from* π*. Otherwise it returns 0 (reject).*

We assume this scheme is executed by a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, both of which are probabilistic polynomial time interactive algorithms. The public transcript generated by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs s and t is denoted by $tr \leftarrow \langle \mathcal{P}(s), \mathcal{V}(t) \rangle$ and we write $\langle \mathcal{P}(s), \mathcal{V}(t) \rangle = b$, $b \in \{0, 1\}$ for rejection or acceptance. Let R be a polynomial time decidable ternary relation. We denote w as a witness for the statement x if $(\sigma, x, w) \in R$ and we define the languages

$$L_\sigma = \{x \mid \exists w : (\sigma, x, w) \in R\}$$

as the set of statements x having a witness w for the relation R.

The triple $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called an *argument* for a relation R with perfect completeness if for all non-uniform polynomial time interactive adversaries $\mathcal{A}$ we have:

(Computational soundness)

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : x \leftarrow \mathcal{A}(\sigma, \text{hist}) : x \notin L_\sigma \wedge \langle \mathcal{A}, \mathcal{V}(\sigma, x) \rangle = 1] \approx 0$$

(Perfect completeness)

$$\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : x \leftarrow \mathcal{A}(\sigma, \text{hist}) : (\sigma, x, w) \notin R \vee \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x) \rangle = 1] = 1$$

Moreover, an argument $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called *public coin* if the verifier chooses their messages uniformly at random and independently of the messages sent by the prover, *i.e.*, the challenges correspond to the verifier's randomness ρ. Then we define:

(Perfect special honest verifier zero knowledge) A public coin argument $(\mathsf{Shuf.Setup}, \mathcal{P}, \mathcal{V})$ is called *perfect special honest verifier zero knowledge* (SHVZK) argument for R with common reference string generator $\mathsf{Shuf.Setup}$ if there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have

$$\begin{aligned} &\Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : (x, w\rho); \\ tr \leftarrow \langle \mathcal{P}(\sigma, x, w), \mathcal{V}(\sigma, x, \rho) \rangle &: (\sigma, x, w) \in R \wedge \mathcal{A}(tr) = 1] \\ &= \Pr[(\sigma, \text{hist}) \leftarrow \mathsf{Shuf.Setup}(1^\lambda) : (x, w\rho); \\ &tr \leftarrow \mathcal{S}(\sigma, x, \rho) : (\sigma, x, w) \in R \wedge \mathcal{A}(tr) = 1] \end{aligned}$$

Then, to construct a fully zero-knowledge argument secure against arbitrary verifiers one can first construct a perfect special honest verifier zero knowledge argument and then convert it into a fully zero-knowledge argument [100].

A scheme with the above properties can be efficiently realized from the proof of shuffle correctness for ciphertexts of [23]. In our setting, we view an ElGamal ciphertext as a commitment (since it is unconditionally binding and computationally hiding) and use proofs of commitment shuffle correctness to convince a verifier that two distinct sets of commitments yield the same set of openings.

2.5 Ideal functionalities

2.5.1 Ideal functionality $\mathcal{F}_{BB}$

The bulletin functionality $\mathcal{F}_{BB}$ is defined in Figure 2.2. It is modified from [64, Fig. 17] to not be authenticated (since we assume anonymous posting to the bulletin board). In Chapter 5 we often omit specifying the MID for brevity.

2.5.2 Ideal functionality $\mathcal{F}_{PKI}$

Next, we present the $\mathcal{F}_{PKI}$ functionality, defined in Figure 2.3, which is taken from [143, Figure 3] with a slight change of wording.

$\mathcal{F}_{BB}$ keeps an initially empty ordered list $\mathcal{M}$ and interacts with a set of parties $\mathcal{P}$ as follows:

Post to Bulletin Board: Upon receiving a message $(\text{POST}, sid, MID, m)$ from a party $\mathcal{P}_i \in \mathcal{P}$, if there is no message $(sid, MID, m') \in \mathcal{M}$, append (sid, MID, m) to the list $\mathcal{M}$ of messages that were posted in the public bulletin board. Then send $(\text{POSTED}, sid, MID, m)$ to $\mathcal{S}$.

Read from Bulletin Board: Upon receiving a message (READ, sid) from a party in $\mathcal{P}$, return $(\text{READ}, sid, \mathcal{M})$ to the caller.

Figure 2.2: Ideal functionality $\mathcal{F}_{BB}$.

$\mathcal{F}_{PKI}$ keeps an initially empty list $\mathcal{M}$ of messages and interacts with a set of parties $\mathcal{P}$ as follows:

Report Query: Upon receiving a message (REPORT, sid, v) from party $\mathcal{P}_i$, record $(\mathcal{P}_i, v)$ in $\mathcal{M}$ iff $\mathcal{P}_i$ does not have a record and no record containing v exists.

Retrieve Query: Upon receiving a message $(\text{RETRIEVE}, sid, \mathcal{P}_i)$, look up and reply with $(\mathcal{P}_i, v)$, where $v = \perp$ when there is no record for $\mathcal{P}_i$.

Figure 2.3: Ideal functionality $\mathcal{F}_{PKI}$.

2.5.3 Ideal functionality $\mathcal{F}_{ZK}$

The $\mathcal{F}_{ZK}$ functionality in Figure 2.4 is adapted from [61, Fig. 7], so that the identity of the prover is not revealed to the verifier. In practice, this can be realized by communicating over a sender-anonymous channel.

$\mathcal{F}_{ZK}$ proceeds as follows, running with the parties in $\mathcal{P}$, and parameterized with a relation R:

Prove: Upon receiving $(\text{ZK-PROVER}, sid, \mathcal{P}_j, x, w)$ from $\mathcal{P}_i$: If $R(x, w) = 1$, then send the message $(\text{ZK-PROOF}, sid, x)$ to $\mathcal{P}_j$ and $\mathcal{S}$. Otherwise, ignore.

Figure 2.4: Ideal functionality $\mathcal{F}_{ZK}$.

2.5.4 Ideal functionality $\mathcal{F}_{NIZK}$

Finally, we give the non-interactive zero knowledge functionality $\mathcal{F}_{NIZK}$ in Figure 2.5, taken from [115, Fig. 4].

Parameterized with relation R and running with the parties in $\mathcal{P}$.

Proof: On input $(\text{PROVE}, sid, x, w)$ from party $\mathcal{P}_i$ ignore if $(x, w) \notin R$. Send (PROVE, sid, x) to $\mathcal{S}$ and wait for answer (PROOF, sid, π). Upon receiving the answer store (x, π) and send (PROOF, sid, π) to $\mathcal{P}_i$.

Verification: On input $(\text{VERIFY}, sid, x, \pi)$ from $\mathcal{P}_j$ check whether (x, π) is stored. If not send $(\text{VERIFY}, sid, x, \pi)$ to $\mathcal{S}$ and wait for an answer $(\text{WITNESS}, sid, w)$. Upon receiving the answer, check whether $(x, w) \in R$ and in that case, store (x, π). If (x, π) has been stored return $(\text{VERIFICATION}, sid, 1)$ to $\mathcal{P}_j$, else return $(\text{VERIFICATION}, sid, 0)$.

Figure 2.5: Ideal functionality $\mathcal{F}_{NIZK}$.

2.6 Blockchain

2.6.1 Simplified UTXO model

In order to focus on the novel aspects of our protocol, we represent cryptocurrency transactions under a simplified version of the Bitcoin UTXO model [149]. For the sake of simplicity we only consider operations of the "Pay to Public Key" (P2PK) output type, which we later show how to realize while keeping the values of transactions private. The formal description of the adopted simplified UTXO model is discussed below.

Bitcoin [149] and other similar cryptocurrencies use the concept of *unspent transaction output*, or UTXO, that represents an indivisible amount of currency locked to an owner [40]. Each transaction contains a certain number of consumed UTXO, named transactions inputs, and created UTXO, named transaction outputs. In particular, a UTXO defines a number representing a certain amount of currency and a *locking script* specifying the conditions that have to be satisfied to use the UTXO as a transaction input (*i.e.*, spend the UTXO). Note that each created UTXO can have a different recipient and that, in case the amount of currency that has to be transferred is smaller than the sum of the inputs, a *change* UTXO is created, *i.e.*, a UTXO that is still owned by the sender of the transaction. Miners have the role of checking if the unlocking conditions are satisfied and if the sum of the inputs is greater than the sum of the outputs. The model is formally defined as follows:

- **Representing Addresses:** An address $Addr = \mathsf{pk}$ is simply a signature verification key associated to a certain secret key sk, where pk and sk are generated by the key generation algorithm $\mathsf{Sig.KeyGen}(1^\lambda)$ and subsequent signatures σ are generated by the signature algorithm $\text{Sig}(\mathsf{sk}, .) = \mathsf{Sig.Sign}_{\mathsf{sk}}(.)$;

- **Representing Transactions:** We represent a transaction in our simplified UTXO model by the tuple $\mathsf{tx} = (\mathtt{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig})$, where $\mathtt{id} \in \{0,1\}^\lambda$ is a unique transaction identification, $\mathsf{In} = \{(\mathtt{id}_1, \mathtt{in}_1), \ldots, (\mathtt{id}_m, \mathtt{in}_m)\}$

is a set of pairs of previous transaction id's $\mathtt{id} \in \{0,1\}^\lambda$ and their values $\mathtt{in} \in \mathbb{N}$, $\mathsf{Out} = \{(\mathtt{out}_1, Addr_1), \ldots, (\mathtt{out}_n, Addr_n)\}$ is a set of pairs of values $\mathtt{out} \in \mathbb{N}$ and addresses $Addr$ and $\mathsf{Sig} = \{\sigma_1, \ldots, \sigma_m\}$ is a set of signatures σ.

- **Transaction Validity:** A transaction $\mathsf{tx} = (\mathtt{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig})$ is considered valid if, for all $(\mathtt{id}_i, \mathtt{in}_i) \in \mathsf{In}$ and $(\mathtt{out}_j, Addr_j) \in \mathsf{Out}$, the following conditions hold:

  1. There exists a valid transaction $\mathsf{tx}_i = (\mathtt{id}_i, \mathsf{In}_i, \mathsf{Out}_i, \mathsf{Sig}_i)$ in the public ledger such that $(\mathtt{in}_i, Addr_i) \in \mathsf{Out}_i$.
  2. There exists $\sigma_i \in \mathsf{Sig}$ such that σ_i is a valid signature of $\mathtt{id}|\mathsf{In}|\mathsf{Out}$ under $Addr_i$, *i.e.*, $Ver(\mathsf{pk}_i, \mathtt{id}|\mathsf{In}|\mathsf{Out}, \sigma_i) = True$.
  3. It holds that $\sum_{i=1}^{m} \mathtt{in}_i = \sum_{j=1}^{n} \mathtt{out}_j$.

- **Generating Transactions:** A party controlling the corresponding signing keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_m$ for valid UTXO addresses $Addr_1, \ldots, Addr_m$ containing values $\mathtt{in}_1, \ldots, \mathtt{in}_m$ can generate a transaction that transfers the funds in these addresses to output addresses $Addr_{out,1}, \ldots, Addr_{out,n}$ by proceeding as follows:

  1. Choose a unique $\mathtt{id} \in \{0,1\}^\lambda$.
  2. Choose values $\mathtt{out}_1, \ldots, \mathtt{out}_n$ such that $\sum_{i=1}^{m} \mathtt{in}_i = \sum_{j=1}^{n} \mathtt{out}_j$.
  3. Generate sets $\mathtt{in}$ and Out as described above and sign $\mathtt{id}|\mathsf{In}|\mathsf{Out}$ with the signing keys corresponding to $Addr_1, \ldots, Addr_m$, *i.e.*, $\sigma_i = \mathsf{Sig.Sign}_{\mathsf{sk}_i}(\mathtt{id}|\mathsf{In}|\mathsf{Out})$ for $i = 1, \cdots, m$, obtaining $\mathsf{Sig} = \{\sigma_1, \ldots, \sigma_m\}$.
  4. Output $\mathsf{tx} = (\mathtt{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig})$.

2.6.2 Confidential transactions

In the case of confidential transactions [144] the input and output amounts are kept secret using Pedersen commitments. However, in order to achieve public verifiability, the transactions contain a zero-knowledge proof that the sum of the inputs is equal to the sum of the outputs, and that all the outputs are between $[0, 2^l - 1]$ (which can be computed with Bullet Proofs [48]). In particular, confidential transactions can be formally defined by modifying the simplified UTXO model described above as follows:

- **Representing inputs and outputs:** Set In is defined as $\mathsf{In} = \{(\mathtt{id}_1, \mathsf{com}(\mathtt{in}_1, r_{\mathtt{in}_1})), \ldots, (\mathtt{id}_m, \mathsf{com}(\mathtt{in}_m, r_{\mathtt{in}_m}))\}$ and set Out is defined as $\mathsf{Out} = \{(\mathsf{com}(\mathtt{out}_1, r_{\mathtt{out}_1}), Addr_1), \ldots, (\mathsf{com}(\mathtt{out}_n, r_{\mathtt{out}_n}), Addr_n)\}$.

- **Generate Transaction with $\mathsf{In}, \mathsf{Out}$:** Compute $\frac{\prod_{j=1}^{n} \mathsf{com}(\mathtt{out}_j, r_{\mathtt{out}_j})}{\prod_{i=1}^{m} \mathsf{com}(\mathtt{in}_i, r_{\mathtt{in}_i})} = \mathsf{com}(0, \sum_{j=1}^{n} r_{\mathtt{out}_j} - \sum_{i=1}^{m} r_{\mathtt{in}_i})$ with $r_{\mathtt{in}_i}, r_{\mathtt{out}_j} \xleftarrow{\$} \mathbb{Z}_q$, include in the transaction

the randomness $\sum_{j=1}^{n} r_{\mathtt{out}_j} - \sum_{i=1}^{m} r_{\mathtt{in}_i}$ and the range proofs π guaranteeing that $\mathtt{out}_1, \cdots, \mathtt{out}_n$ are between $[0, 2^l - 1]$. The resulting transaction is then represented by $\mathsf{tx} = (\mathtt{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig}, \sum_{j=1}^{n} r_{\mathtt{out}_j} - \sum_{i=1}^{m} r_{\mathtt{in}_i}, \pi)$.

- **Validate a Transaction** tx**:** Compute $\frac{\prod_{j=1}^{n} \mathsf{com}(\mathtt{out}_j, r_{\mathtt{out}_j})}{\prod_{i=1}^{m} \mathsf{com}(\mathtt{in}_i, r_{\mathtt{in}_i})} = \mathsf{com}(s, t)$ and check if the obtained commitments is equal to $\mathsf{com}(0, \sum_{j=1}^{n} r_{\mathtt{out}_j} - \sum_{i=1}^{m} r_{\mathtt{in}_i})$, guaranteeing that $\sum_{i=1}^{m} \mathtt{in}_i = \sum_{j=1}^{n} \mathtt{out}_j$, then check the validity of the range proofs π.

- **Spend a transaction output** Out**:** Parse $\mathsf{Out} = (\mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i}), Addr_i)$. In order to spend Out, the commitment $\mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i}) = g^{\mathtt{out}_i} h^{r_{\mathtt{out}_i}}$ has to be opened by revealing $\mathtt{out}_i$ and $r_{\mathtt{out}_i}$. Values $\mathtt{out}_i$ and $r_{\mathtt{out}_i}$ are included in a regular UTXO transaction generated as described above. Later on, this UTXO transaction can be validated by checking that $\mathtt{out}_i, r_{\mathtt{out}_i}$ is a valid opening of $\mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i})$ and following the steps of a regular UTXO transaction validation.

- **Spend a transaction output** Out **with a NIZKPoK of** $r_{\mathtt{out}_i}$**:** Alternatively, an output $\mathsf{Out} = (\mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i}), Addr_i)$ for which only $\mathtt{out}_i$ and $\hat{h} = h^{r_{\mathtt{out}_i}}$ (but not $r_{\mathtt{out}_i}$) are known can be spent if a NIZK π' proving knowledge of $r_{\mathtt{out}_i}$ is also available. Notice that knowing $\mathtt{out}_i$ is sufficient for validating the regular UTXO transaction created using Out as an input. Moreover, it can be checked that $g^{\mathtt{out}_i} h^{r_{\mathtt{out}_i}} = \mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i})$ given $\mathtt{out}_i$ and $\hat{h} = h^{r_{\mathtt{out}_i}}$, while the proof π' guarantees that $\hat{h} = h^{r_{\mathtt{out}_i}}$ is well formed[1]. Values $\mathtt{out}_i$, $h^{r_{\mathtt{out}_i}}$ and the proof π' are included in a regular UTXO transaction generated as described above. Later on, this UTXO transaction can be validated by checking that $g^{\mathtt{out}_i} h^{r_{\mathtt{out}_i}} = \mathsf{com}(\mathtt{out}_i, r_{\mathtt{out}_i})$, checking that π' is valid and following the steps of a regular UTXO transaction validation.

Note that the input set In in confidential transactions can also be public, (*i.e.*, $\mathsf{In} = \{(\mathtt{id}_1, \mathtt{in}_1), \ldots, (\mathtt{id}_m, \mathtt{in}_m)\}$), as long as the outputs are kept private.

[1] In fact, showing such a proof of knowledge π' of $r_{\mathtt{out}_i}$ together with $h^{r_{\mathtt{out}_i}}$ and $\mathtt{out}_i$ makes it easy to adapt reduction of the binding property of the Pedersen commitment scheme to the Discrete Logarithm assumption. Instead of obtaining $r_{\mathtt{out}_i}$ from the adversary, the reduction simply extracts it from π'.

Chapter 3

FAST: Fair Auctions via Secret Transactions

In this Chapter, we present the results published in the 20th International Conference on Applied Cryptography and Network Security (ACNS 2022) [86] and available on Cryptology ePrint Archive [85].

We propose efficient privacy-preserving protocols, *i.e.*, with no need of a trusted third party, for both first and second-price sealed-bid auctions (Sections 3.6 and 3.7) with *fairness* against rational adversaries, leveraging secret cryptocurrency transactions and public smart contracts. In our approach, it is ensured that cheaters are *identified* and *financially punished* by losing a *secret collateral deposit* (Sections 3.5). Indeed, it is always more profitable to execute the protocol honestly than to cheat (Section 3.9).

3.1 Our Techniques

We start with a first-price sealed-bid auction protocol that builds on a simple passively secure protocol similar to that of SEAL [13] and compile it to achieve active security. However, we not only obtain an actively secure protocol but also add cheater identification and public verifiability properties. We use these properties to add our financial punishment mechanism with secret deposits to this protocol. Even though our protocol achieves stronger security guarantees than SEAL (*i.e.*, sequential composability and fairness guarantees), it is more efficient than the SEAL protocol as shown in Section 3.8. Later on, we extend this protocol to the second-price case with a very low performance overhead.

A Toy Example:

Our protocol uses a modified version of the *Anonymous Veto Protocol* from [117] as a building block. The anonymous veto protocol allows a set of n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ to anonymously indicate whether they want to veto or not on a

particular subject by essentially securely computing the logical-OR function of their inputs. In this protocol, each party $\mathcal{P}_i$ has an input bit $d_i \in \{0,1\}$ with 0 indicating no veto and 1 indicating veto, and they wish to compute $\bigvee_{i=1}^{n} d_i$. As proposed in [13], this simple anonymous veto protocol can be used for auctions by having parties evaluate their bids bit-by-bit, starting from the most significant bit and proceeding to execute the veto protocol for each bit in the following way: 1. Until there is no veto, all parties only veto (input $d_i = 1$ in the veto protocol) if and only if the current bit of their bid is 1; 2. After the first veto, a party only vetoes if the bit of her bid in the last time a veto happened was 1 *and* the current bit is also 1. In other words, in this toy protocol, parties stop vetoing once they realize that there is another party with a higher bid (*i.e.*, there was a veto in a round when their own bit were 0) and the party with the highest bid continues vetoing according to her bid until the last bit. Therefore, the veto protocol output represents the highest bid. However, a malicious party can choose not to follow the protocol, altering the output.

Achieving Active Security with Cheater Identification and Public Verifiability:

In order to achieve active security with cheater identification and public verifiability we depart from a simple passively secure protocol and compile it into an active secure protocol using NIZKs following an approach similar to that of [108, 121, 127]. This ensures that at every round of the protocol all parties' inputs are computed according to the protocol rules, including previous rounds' inputs and outputs. However, since the generic techniques from [108, 121, 127] yield highly inefficient protocols, we carefully construct tailor-made efficient non-interactive zero-knowledge proofs for our specific protocol, ensuring it to be efficient.

Incentivizing Correct Behaviour with Secret Deposits:

In order to create incentives for parties to behave honestly, a deposit based on their bids is required. However, a public deposit would leak information about the parties' bids, which have to be kept secret. Hence, we do secret deposits as discussed below and keep the amount secret unless a party is identified as a cheater, in which case the cheater's deposit is distributed among the honest parties. The cheater is then eliminated and the protocol is re-executed with the remaining parties using their initial bids/deposits so that a winner is determined. This makes it rational not to cheat both in the case of first and second-price auctions, *i.e.*, cheating always implies a lower utility than behaving honestly (see Section 3.9).

Achieving On-Chain Efficiency:

In order to minimize the amount of on-chain communication, an approach based on techniques from [18] is adopted. Every time a message is sent from a given

party to the other parties, all of them sign the message received and send the signature to each other. Communication is only done on-chain (through the smart contract) in case of suspected cheating.

Secret Deposits to Public Smart Contracts:

Since we use secret deposits based on confidential transactions [144], we need a mechanism to reveal the value of cheating parties' deposits to the smart contract so it can punish cheaters. We do that by secret sharing trapdoor information used to reveal this value using a publicly verifiable secret sharing (PVSS) scheme [64] that allows us to prove in zero-knowledge both that the shares are valid and that they contain the trapdoor for a given deposit. These shares are held by a committee that does not act unless cheating is detected, in which case the committee members are reimbursed for reconstructing the trapdoor with funds from the cheater's deposit itself. We discuss this approach in Section 3.5. Providing alternative methods for holding these deposits is an important open problem.

3.2 Security Model and Setup Assumptions

We prove our protocol secure in the real/ideal simulation paradigm with sequential composition. Our protocol uses the Random Oracle Model (ROM) [27]. Note that adopting the UC model, as an alternative, requires to use UC-secure NIZK (instead of those described subsequently), but reduces the efficiency of the protocol. Also, previous works consider the sequential composability model only.

Adversarial Model:

We consider *malicious* adversaries that may deviate from the protocol in any arbitrary way. Moreover, we consider the *static* case, where the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. Moreover, we assume that parties have access to synchronous communication channels, *i.e.*, all messages are delivered within a given round with a known maximum delay.

3.3 Non-interactive Zero-Knowledge Proofs of Knowledge

In our work we follow the approach of Camenisch and Stadler [58] based on the Fiat-Shamir heuristic [94] in order to obtain non interactive zero knowledge (NIZK) proofs of knowledge for discrete logarithm relations. We will use these NIZKs in forcing parties to execute our protocols correctly using the GMW methodology [108]. Notice that parties must provide such NIZKs proving they have executed each round of the protocol correctly, and an invalid NIZK is also

a publicly verifiable proof that the party has misbehaved. We will be using the following NIZKs:

NIZK for Stage 2 - Before First Veto:

In Stage 2 of the protocol, we need a NIZK proving knowledge of either b_{ir}, r_{ir}, x_{ir} such that $\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}$ or of $b_{ir}, r_{ir}, \bar{r}_{ir}$ such that $\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge v_{ir} = g^{\bar{r}_{ir}}$, where $v_{ir}, c_{ir}, X_{ir}, g, Y_{ir}$ are public. We denote this NIZK by

$$BV\{b_{ir}, r_{ir}, x_{ir}, \bar{r}_{ir} \mid (\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}) \vee (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge v_{ir} = g^{\bar{r}_{ir}})\}.$$

A detailed construction of this NIZK is discussed below.

Following the approach proposed by Camenisch and Stadler [58] we construct this NIZK as follows:

$$\tilde{F}_1 = DL(h, c_{ir}) \otimes \Big[DL(Y_{ir}, v_{ir}) \cap DL(g, X_{ir})\Big]$$

$$\tilde{F}_2 = DL(h, c_{ir}/g) \otimes DL(g, v_{ir})$$

Therefore we need to show

$$\tilde{F} = \tilde{F}_1 \cup \tilde{F}_2$$

To prove the knowledge of either $\tilde{F}_1$ or $\tilde{F}_2$, assuming that $\tilde{F}_\alpha$ is known, party i proceeds as follows:

1. Choose $\bar{V} = (\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4)$ with $\bar{v}_i \xleftarrow{\$} \mathbb{Z}_q$ and $\bar{w} = (\bar{w}_1, \bar{w}_2)$ with $\bar{w}_\alpha = 0$ and $\bar{w}_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \neq \alpha$, and compute $t_1 = c_{ir}^{\bar{w}_1} h^{\bar{v}_1}$, $t_2 = v_{ir}^{\bar{w}_1} Y_{ir}^{\bar{v}_2}$, $t_3 = X_{ir}^{\bar{w}_1} g^{\bar{v}_2}$, $t_4 = (\frac{c_{ir}}{g})^{\bar{w}_2} h^{\bar{v}_3}$, and $t_5 = v_{ir}^{\bar{w}_2} g^{\bar{v}_4}$

2. $H = \mathcal{H}(h, c_{ir}, Y_{ir}, v_{ir}, g, X_{ir}, \frac{c_{ir}}{g}, t_1, t_2, t_3, t_4, t_5) \pmod q$.

3. $\Gamma = (\gamma_1, \gamma_2)$ where

$$\gamma_i = \begin{cases} H - (\bar{w}_1 + \bar{w}_2) \pmod q, & \textit{if} \quad i = \alpha \\ \bar{w}_i, & \textit{otherwise} \end{cases}$$

4. $R = (r_1, r_2, r_3, r_4, r_5)$ where $r_1 = \bar{v}_1 - \gamma_\alpha x_1$, $r_2 = \bar{v}_2 - \gamma_\alpha x_2$, $r_3 = \bar{v}_2 - \gamma_\alpha x_2$, $r_4 = \bar{v}_3 - \gamma_\alpha x_3$, and $r_5 = \bar{v}_4 - \gamma_\alpha x_4$ (all equations are modulo q), in which $(x_1, x_2, x_3, x_4) = (r_{ir}, x_{ir}, 0, 0)$ if $\alpha = 1$, and $(x_1, x_2, x_3, x_4) = (0, 0, r_{ir}, \bar{r}_{ir})$ if $\alpha = 2$.

The resulting proof is $(\gamma_1, \gamma_2, r_1, r_2, r_3, r_4, r_5)$. The validity of the proof can be checked by first re-constructing the commitments. That is,

$$t'_1 = c_{ir}^{\gamma_1} h^{r_1} \qquad t'_2 = v_{ir}^{\gamma_1} Y^{r_2} \qquad t'_3 = X_{ir}^{\gamma_1} g^{r_3}$$
$$t'_4 = (\frac{c_{ir}}{g})^{\gamma_2} h^{r_4} \qquad t'_5 = v_{ir}^{\gamma_2} g^{r_5}$$

and then checks the following condition

$$\gamma_1 + \gamma_2 = \mathcal{H}(h, c_{ir}, Y_{ir}, v_{ir}, g, X_{ir}, \frac{c_{ir}}{g}, t'_1, t'_2, t'_3, t'_4, t'_5)$$

NIZK for Stage 3 - After First Veto:

In Stage 3 of the protocol, we need a NIZK proving knowledge of either b_{ir}, r_{ir}, x_{ir} such that $\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}$, or of $b_{ir}, r_{ir}, \bar{r}_{i\hat{r}}, \bar{r}_{ir}$ such that $\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge d_{i\hat{r}} = g^{\bar{r}_{i\hat{r}}} \wedge v_{ir} = g^{\bar{r}_{ir}}$, or of $b_{ir}, r_{ir}, x_{i\hat{r}}, x_{ir}$ such that $\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge \ d_{i\hat{r}} = Y_{i\hat{r}}^{x_{i\hat{r}}} \wedge X_{i\hat{r}} = g^{x_{i\hat{r}}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}$. We denote this NIZK by

$$\begin{aligned} AV\{b_{ir}, r_{ir}, x_{ir}, \bar{r}_{i\hat{r}}, \bar{r}_{ir}, x_{i\hat{r}} \,|\, & (\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}) \vee \\ & (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge d_{i\hat{r}} = g^{\bar{r}_{i\hat{r}}} \wedge v_{ir} = g^{\bar{r}_{ir}}) \vee \\ & (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge \ d_{i\hat{r}} = Y_{i\hat{r}}^{x_{i\hat{r}}} \wedge X_{i\hat{r}} = g^{x_{i\hat{r}}} \wedge \\ & v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}})\}. \end{aligned}$$

A detailed construction of this NIZK is discussed below.

Following the approach proposed by Camenisch and Stadler [58] we construct this NIZK as follows:

$$\tilde{F}_1 = DL(h, c_{ir}) \otimes \Big[DL(Y_{ir}, v_{ir}) \cap DL(g, X_{ir})\Big]$$

$$\tilde{F}_2 = DL(h, c_{ir}/g) \otimes DL(g, d_{i\hat{r}}) \otimes DL(g, v_{ir})$$

$$\tilde{F}_3 = DL(h, c_{ir}/g) \otimes \Big[DL(Y_{i\hat{r}}, d_{i\hat{r}}) \cap DL(g, X_{i\hat{r}})\Big] \otimes \Big[DL(Y_{ir}, v_{ir}) \cap DL(g, X_{ir})\Big]$$

Therefore we need to show

$$\tilde{F} = \tilde{F}_1 \cup \tilde{F}_2 \cup \tilde{F}_3$$

To prove the knowledge of either $\tilde{F}_1$ or $\tilde{F}_2$ or $\tilde{F}_3$, assuming that $\tilde{F}_\alpha$ is known, party i proceeds as follows:

1. Choose $\bar{V} = (\bar{v}_1, \bar{v}_2, \bar{v}_3, \bar{v}_4, \bar{v}_5, \bar{v}_6, \bar{v}_7, \bar{v}_8)$ with $\bar{v}_i \xleftarrow{\$} \mathbb{Z}_q$ and $\bar{w} = (\bar{w}_1, \bar{w}_2, \bar{w}_3)$ with $\bar{w}_\alpha = 0$ and $\bar{w}_i \xleftarrow{\$} \mathbb{Z}_q$ for $i \neq \alpha$, and compute $t_1 = c_{ir}^{\bar{w}_1} h^{\bar{v}_1}$, $t_2 = v_{ir}^{\bar{w}_1} Y_{ir}^{\bar{v}_2}$, $t_3 = X_{ir}^{\bar{w}_1} g^{\bar{v}_2}$, $t_4 = (\frac{c_{ir}}{g})^{\bar{w}_2} h^{\bar{v}_3}$, $t_5 = d_{i\hat{r}}^{\bar{w}_2} g^{\bar{v}_4}$, $t_6 = v_{ir}^{\bar{w}_2} g^{\bar{v}_5}$, $t_7 = (\frac{c_{ir}}{g})^{\bar{w}_3} h^{\bar{v}_6}$, $t_8 = d_{i\hat{r}}^{\bar{w}_3} Y_{i\hat{r}}^{\bar{v}_7}$, $t_9 = X_{i\hat{r}}^{\bar{w}_3} g^{\bar{v}_7}$, $t_{10} = v_{ir}^{\bar{w}_3} Y_{ir}^{\bar{v}_8}$ and $t_{11} = X_{ir}^{\bar{w}_3} g^{\bar{v}_8}$.

2. $H = \mathcal{H}(h, c_{ir}, Y_{ir}, v_{ir}, g, X_{ir}, \frac{c_{ir}}{g}, d_{i\hat{r}}, Y_{i\hat{r}}, X_{i\hat{r}}, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11})$.

3. $\Gamma = (\gamma_1, \gamma_2, \gamma_3)$ where

$$\gamma_i = \begin{cases} H - (\bar{w}_1 + \bar{w}_2 + \bar{w}_3) \pmod q, & \text{if} \quad i = \alpha \\ \bar{w}_i, & \text{otherwise} \end{cases}$$

4. $R = (r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11})$ where $r_1 = \bar{v}_1 - \gamma_\alpha x_1$, $r_2 = \bar{v}_2 - \gamma_\alpha x_2$, $r_3 = \bar{v}_2 - \gamma_\alpha x_2$, $r_4 = \bar{v}_3 - \gamma_\alpha x_3$, $r_5 = \bar{v}_4 - \gamma_\alpha x_4$, $r_6 = \bar{v}_5 - \gamma_\alpha x_5$, $r_7 = \bar{v}_6 - \gamma_\alpha x_6$, $r_8 = \bar{v}_7 - \gamma_\alpha x_7$, $r_9 = \bar{v}_7 - \gamma_\alpha x_7$, $r_{10} = \bar{v}_8 - \gamma_\alpha x_8$ and $r_{11} = \bar{v}_8 - \gamma_\alpha x_8$ (all equations are modulo q), in which $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (r_{ir}, x_{ir}, 0, 0, 0, 0, 0, 0)$ if $\alpha = 1$, and $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (0, 0, r_{ir}, \bar{r}_{i\hat{r}}, \bar{r}_{ir}, 0, 0, 0)$ if $\alpha = 2$, and $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (0, 0, 0, 0, 0, r_{ir}, x_{i\hat{r}}, x_{ir})$ if $\alpha = 3$.

The resulting proof is $(\gamma_1, \gamma_2, \gamma_3, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11})$. The validity of the proof can be checked by first re-constructing the commitments. That is,

$$\begin{aligned} t'_1 &= c_{ir}^{\gamma_1} h^{r_1} & t'_2 &= v_{ir}^{\gamma_1} Y^{r_2} & t'_3 &= X_{ir}^{\gamma_1} g^{r_3} \\ t'_4 &= (\frac{c_{ir}}{g})^{\gamma_2} h^{r_4} & t'_5 &= d_{i\hat{r}}^{\gamma_2} g^{r_5} & t_6 &= v_{ir}^{\gamma_2} g^{r_6} \\ t'_7 &= (\frac{c_{ir}}{g})^{\gamma_3} h^{r_7} & t'_8 &= d_{i\hat{r}}^{\gamma_3} Y_{i\hat{r}}^{r_8} & t'_9 &= X_{i\hat{r}}^{\gamma_3} g^{r_9} \\ t'_{10} &= v_{ir}^{\gamma_3} Y_{ir}^{r_{10}} & t'_{11} &= X_{ir}^{\gamma_3} g^{r_{11}} \end{aligned}$$

and then checks the following condition

$$\gamma_1 + \gamma_2 + \gamma_3 = \mathcal{H}(h, c_{ir}, Y_{ir}, v_{ir}, g, X_{ir}, \frac{c_{ir}}{g}, d_{i\hat{r}}, Y_{i\hat{r}}, X_{i\hat{r}}, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11})$$

NIZK for Recovery - Proof of Not Winning:

In case the winning bid b_w is determined but the winner $\mathcal{P}_w$ does not reveal herself, the honest parties in our auction protocol will prove in zero-knowledge that they are not the winner. In order to do so, they use a NIZK $NW_i \leftarrow NW\{x_{i1}, \ldots, x_{il} \mid (V_1 = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}}) \vee \ldots \vee (V_l = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}})\}$ that can be constructed using the techniques from [58]. We do not describe this NIZK construction nor estimate its complexity because it is used only in a corner case of cheating where the honest parties are reimbursed for generating and verifying such proofs.

NIZK for PVSS share consistency CC:

As part of our secret deposit mechanism (further discussed in Section 3.5), we will use a NIZK showing that shares computed with the PVSS protocol π_{PVSS} from [64] encode secrets g^m and h^r that are terms of a Pedersen commitment $c = g^m h^r$. Formally, given generators $g_1, \ldots, g_n, g, h$ of a cyclic group $\mathbb{G}_q$ of prime order q, pairwise distinct elements $\alpha_1, \ldots, \alpha_n$ in $\mathbb{Z}_q$ and a Pedersen commitment $c = g^m h^r$ known by prover and verifier, for $p(x)$ and m, r known by the prover, this NIZK is used to prove that $(\hat{\sigma}_1, \ldots, \hat{\sigma}_n) \in \{(g_1^{p(\alpha_1)}, \ldots, g_n^{p(\alpha_n)}) : p \in \mathbb{Z}_q[X], p(-1) = g^m, p(-2) = h^r\}$. We denote this NIZK by $CC((g_i)_{i\in[n]}, (\alpha_i)_{i\in[n]}, g, h, c, (\hat{\sigma}_i)_{i\in[n]})$. Notice that this NIZK can be constructed using the techniques from [58] and integrated with the NIZK $LDEI$ already used in π_{PVSS} (presented above).

3.4 Modelling a Stateful Smart Contract

We employ a stateful smart contract functionality $\mathcal{F}_{\mathsf{SC}}$ similar to that of [83] in order to model the smart contract that implements the financial punishment mechanism for our protocol. For the sake of simplicity, we assume that each instance of $\mathcal{F}_{\mathsf{SC}}$ is already parameterized by the address of the auctioneer party who will receive the payment for the auctioned good, as well as by the identities (and public keys) of the parties in a secret deposit committee $\mathcal{C}$ that will help the smart contract to open secret deposits given by parties in case cheating is detected. We also assume that $\mathcal{F}_{\mathsf{SC}}$ has a protocol verification mechanism pv for verifying the validity of protocol messages. $\mathcal{F}_{\mathsf{SC}}$ is described in Figures 3.1 and 3.2.

3.5 Secret Deposits in Public Smart Contracts

When using secret deposits as in our application, it is implied that there exists a secret trapdoor that can be used to reveal the value of such deposits (and transfer them). However, since we base our financial punishment mechanism on a standard public smart contract, we cannot expose the trapdoor to the smart contract. Instead, we propose that a committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ with $m/2+2$ honest members[1] holds this trapdoor in a secret shared form. This committee does not act unless a cheating party needs to be punished and the trapdoor needs to be reconstructed in order to allow the smart contract to transfer her collateral deposit. In this case, the committee itself can be reimbursed from the collateral funds. We present a practical construction following this approach. Proposing more methods for keeping custody of such secret deposits is left as an important open problem.

[1] We need $m/2 + 2$ honest members to instantiate our packed publicly verifiable secret sharing based solution where two group elements are secret shared with a single share vector.

Functionality $\mathcal{F}_{\mathsf{SC}}$ (Stages 1,2,3 and 4)

$\mathcal{F}_{\mathsf{SC}}$ is executed with parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, a deposit committee members $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and parameterized by committee public keys $\mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m}$ used to run π_{PVSS} and $\mathsf{pk}'_{\mathcal{C}_1}, \ldots, \mathsf{pk}'_{\mathcal{C}_m}$ used for signatures. $\mathcal{F}_{\mathsf{SC}}$ is also parameterized by its own address $Addr_s$, the address of the auctioneer $Addr_{auc}$, timeout limit τ, a protocol verification algorithm pv that given a protocol transcript outputs 1 if it is valid and replies 0 otherwise, and values $work_{\mathcal{P}}$ and $work_{\mathcal{C}}$ for the costs of executing the protocol for the parties in $\mathcal{P}$ and $\mathcal{C}$, respectively. Once execution starts $\mathcal{F}_{\mathsf{SC}}$ samples random generators $g, h \xleftarrow{\$} \mathbb{G}$ for a group $\mathbb{G}$.

- **Stage 1 - Setup:**
  - Upon receiving (PARAM, sid) from $\mathcal{P}_i$, $\mathcal{F}_{\mathsf{SC}}$ returns (PARAM, $sid, g, h, \mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m}$).
  - Upon receiving the message (SETUP, $sid, \mathcal{P}_i, \mathsf{tx}_i, \mathsf{pk}_i, SH1_i|SH2_i$, $Sig_{\mathcal{C}_1,i}, \ldots, Sig_{\mathcal{C}_m,i}$) from $\mathcal{P}_i$, $\mathcal{F}_{\mathsf{SC}}$ checks that this message is valid according to pv. If all checks pass, $\mathcal{F}_{\mathsf{SC}}$ continues the execution. Otherwise, it proceeds to the Recovery Stage otherwise. After receiving this message from all $\mathcal{P}_i \in \mathcal{P}$, $\mathcal{F}_{\mathsf{SC}}$ proceeds to the next steps.
- **Stages 2 and 3:** During these Stages, $\mathcal{F}_{\mathsf{SC}}$ only expects messages in case of suspected cheating, eventually proceeding to the Recovery Stage.
- **Stage 4 - Output:**
  - First-Price Auction: Upon receiving (OUTPUT, $sid, \mathcal{P}_w, b_w, r_{b_w}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]}$) from $\mathcal{P}_w$, $\mathcal{F}_{\mathsf{SC}}$ verifies that b_w, r_{b_w} is a valid opening to the commitment c_w in tx_w and that all signatures $\{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]}$ are valid. If all checks pass, $\mathcal{F}_{\mathsf{SC}}$ performs a transaction towards the auctioneer paying b_w tokens to $Addr_{auc}$, refunds $\mathcal{P}_w$ with $work_{\mathcal{P}}$ tokens and refunds all the other parties $\mathcal{P}_i \in \mathcal{P} \setminus \mathcal{P}_w$ with their full tx_i transaction. Otherwise, proceed to the Recovery Stage.
  - Second-Price Auction: Upon receiving (OUTPUT, $sid, \mathcal{P}_w, \mathsf{tx}_{pay}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(\mathcal{P}_w|b_{w_2})\}_{k\in[n]}$) from $\mathcal{P}_w$, $\mathcal{F}_{\mathsf{SC}}$ verifies that that the range proof π_{pay} is valid and that $\mathsf{com}(b_w, r_{b_w})/\mathsf{com}(change'_w, r_{change'_w}) = \mathsf{com}(b_{w_2}, r_{b_w} - r_{change'_w})$. If the checks pass, then $\mathcal{F}_{\mathsf{SC}}$ refunds all the other parties $\mathcal{P}_i \in \mathcal{P} \setminus \mathcal{P}_w$ with their full tx_i transaction and performs transaction tx_w paying b_{w_2} to the auctioneer and refunding $\mathcal{P}_w$ with $work_{\mathcal{P}}$ tokens. If the check does not pass, then proceed to the Recovery Stage.

Figure 3.1: Functionality $\mathcal{F}_{\mathsf{SC}}$ (Stages 1,2,3 and 4).

A possible solution:

A feasible but not practical approach to do this would be storing the trapdoor with the mechanism proposed in [31], where a secret is kept by obliviously and randomly chosen committees by means of a proactive secret sharing scheme where each current committee "encrypts the secret to the future" in such a way that the next committee can open it. However, it is also necessary to ensure

Functionality $\mathcal{F}_{\mathsf{SC}}$ (Recovery)

- **Recovery:** Upon receiving one of the following messages $\mathcal{F}_{\mathsf{SC}}$ from a party $\mathcal{P}_i \in \mathcal{P}$ acts as described:
  - (RECOVERY-MISSING, $sid, msg, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(msg)\}_{k\in[n]}$): $\mathcal{F}_{\mathsf{SC}}$ sends (REQUEST, sid) to each $\mathcal{P}_i \in \mathcal{P}$. If $\mathcal{P}_i$ does not send the missing message msg_{ri} or $sig_{\mathsf{sk}_i}(msg_{r-1,i'})$ before a timeout τ or sends an invalid message according to pv, it is considered a cheater.
  - (RECOVERY-CHEAT, $sid, \mathcal{P}_j, \pi_c$): in case $\mathcal{P}_j$ has been accused to cheat by sending conflicting messages $msg_{rj} \neq msg'_{rj}$ then $\pi_c = (msg_{rj}, \mathsf{Sig.Sign}_{\mathsf{sk}_j}(msg_{rj}), msg'_{rj}, sig_{\mathsf{sk}_j}(msg'_{rj}))$, while in case $\mathcal{P}_j$ has been accused to cheat by sending an invalid message msg_{rj} according to pv then $\pi_c = (msg_{rj}, \mathsf{Sig.Sign}_{\mathsf{sk}_j}(msg_{rj}))$. If π_c is a valid proof of cheating, then $\mathcal{P}_j$ is identified as a cheater, otherwise the sender $\mathcal{P}_i$ identified as a cheater.
  - (RECOVERY-PAYMENT, sid, NW_i): $\mathcal{F}_{\mathsf{SC}}$ verifies that the proof of not winning NW_i is valid, *i.e.*, the sender $\mathcal{P}_i$ is not the winner. After a timeout τ counted from the moment the first message is received, all parties $\mathcal{P}_k$ who did not send a valid message (RECOVERY-PAYMENT, sid, NW_i) are considered cheaters (*i.e.*, either the corrupted $\mathcal{P}_w$ in case all other parties sent a valid NW_i or all parties who did not collaborate).
  - (RECOVERY-DISHONEST-WINNER, sid, NW_i): *(second-price case only)* $\mathcal{F}_{\mathsf{SC}}$ verifies that the proof of not winning NW_i is valid, *i.e.*, the sender $\mathcal{P}_i$ is not the winner. Then, $\mathcal{F}_{\mathsf{SC}}$ all parties $\mathcal{P}_k$ who did not send a valid message (RECOVERY-DISHONEST-WINNER, sid, NW_i) are considered cheaters.

  If a party $\mathcal{P}_i$ is a cheater, $\mathcal{F}_{\mathsf{SC}}$ collaborates with $\mathcal{C}$ to open $\mathcal{P}_i$'s deposit tx_i by sending (OPEN, $sid, \mathcal{P}_i$) to each $\mathcal{C}_j \in \mathcal{C}$, takes its full deposit, reimburses each $\mathcal{C}_i \in \mathcal{C}$ with $work_{\mathcal{C}}$ tokens and reimburses each party $\mathcal{P}\backslash\mathcal{P}_i$ with an equal share of the remaining tokens. In particular, $\mathcal{F}_{\mathsf{SC}}$ proceeds as follows:

  - Upon receiving (SHARE-DECRYPTION, $sid, (\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i, \tilde{\sigma}_{ij}, DLEQ_{ij}$)) from $\mathcal{C}_j$, $\mathcal{F}_{\mathsf{SC}}$ verifies that $SH2_i = \mathcal{H}((\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i)$ where $SH2_i$ is the one sent from $\mathcal{P}_i$ in Stage 1 - Setup.
  - Use the share decryption verification procedure from π_{PVSS} to identify $2 + n/2$ valid shares $\tilde{\sigma}_{ij}$ (by verifying $DLEQ_{ij}$) and then uses the secret reconstruction algorithm from π_{PVSS} to reconstruct g^{b_i} and $g^{r_{b_i}}$. Next, $\mathcal{F}_{\mathsf{SC}}$ recovers b_i from g^{b_i} (since the length l of the bid is limited).
  - The deposit of the cheating party $\mathcal{P}_i$ is distributed among the honest parties $\mathcal{P} \setminus \mathcal{P}_i$ and $\mathcal{C}$ by spending the confidential transaction output $(c_i, Addr_s)$ of tx_i. Indeed, in order to spend $(c_i, Addr_s)$ it is sufficient to reveal b_i, $h^{r_{b_i}}$ and providing $(\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i, \tilde{\sigma}_{ij}, DLEQ_{ij}$ to prove that $h^{r_{b_i}}$ contains the same r_{b_i} of the initial commitment c_i.

  $\mathcal{F}_{\mathsf{SC}}$ now expects the execution to restart with parties $\mathcal{P} \setminus \mathcal{P}_i$ only.

Figure 3.2: Functionality $\mathcal{F}_{\mathsf{SC}}$ (Recovery).

that the secrets actually correspond to the trapdoor for the parties' deposits. Providing such proofs with the scheme of [31] would require expensive generic zero-knowledge techniques (or a trusted setup for a zk-SNARK).

Protocol $\Pi_{\mathcal{C}}$

Let $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ be the deposit committee members and $\mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m}$ and $\mathsf{sk}_{\mathcal{C}_1}, \ldots, \mathsf{sk}_{\mathcal{C}_m}$ be their public keys and private keys, used to run π_{PVSS}, respectively. Moreover, let $\mathsf{pk}'_{\mathcal{C}_1}, \ldots, \mathsf{pk}'_{\mathcal{C}_m}$ and $\mathsf{sk}'_{\mathcal{C}_1}, \ldots, \mathsf{sk}'_{\mathcal{C}_m}$ be their public keys and private keys, used for signatures, respectively. The following steps are executed by $\mathcal{C}_j \in \mathcal{C}$:

- **Setup verification:** Upon receiving $(\text{SETUP}, sid, \mathcal{P}_i, \mathsf{tx}_i, \mathsf{pk}_i, (\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i)$ from $\mathcal{P}_i$, $\mathcal{C}_j$ checks that tx_i is valid, verifies the shares $(\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im})$ correctness with respect to the committee public keys $\mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m}$ using the verification procedure of π_{PVSS} through $LDEI_i$ and verifies NIZK CC_i. If all the checks pass, compute the hashes $SH1_i = \mathcal{H}(\mathsf{tx}_i, \mathsf{pk}_i)$ and $SH2_i = \mathcal{H}((\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i)$ and the signature $Sig_{\mathcal{C}_j,i} = \mathsf{Sig.Sign}_{\mathsf{sk}'_{\mathcal{C}_j}}(SH1_i | SH2_i)$, then send $(\text{SETUP-VERIFICATION}, sid, Sig_{\mathcal{C}_j,i})$ to $\mathcal{P}_i$.
- **Share decryption:** Upon receiving $(\text{OPEN}, sid, \mathcal{P}_i)$ from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{C}_j$ uses the share decryption procedure from π_{PVSS} on $\hat{\sigma}_{ij}$, obtaining $\tilde{\sigma}_{ij}, DLEQ_{ij}$. and sending $(\text{SHARE-DECRYPTION}, sid, (\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i, \tilde{\sigma}_{ij}, DLEQ_{ij}))$ to $\mathcal{F}_{\mathsf{SC}}$.

Figure 3.3: Protocol $\Pi_{\mathcal{C}}$.

A protocol based on PVSS:

As an alternative, we propose leveraging the structure of our confidential transaction based deposits to secret share their openings with a recent efficient publicly verifiable secret sharing (PVSS) scheme called Albatross [64]. Notice that the secret amount information b_i in these deposits is represented as a Pedersen commitment $g^{b_i}h^{r_i}$ and that the Albatross PVSS scheme also allows for sharing a group element g^s, while proving in zero-knowledge discrete logarithm relations involving g^s in such a way that they can be verified by any third party with access to the public *encrypted* share. Hence, we propose limiting the bid b_i bit length in such a way that we can employ the same trick as in *lifted ElGamal* and have each party $\mathcal{P}_i$ share both g^{b_i} and h^{r_i} with the Albatross PVSS while proving that their public encrypted shares correspond to a secret deposit $g^{b_i}h^{r_i}$. The validity of this claim can be verified by the committee $\mathcal{C}$ itself or the smart contract during Stage 1 - Setup. Later on, if b_i needs to be recovered, $\mathcal{C}$ can reconstruct g^{b_i}, brute force b_i (because it has a restricted bit-length) and deliver it to the smart contract while proving it has been correctly computed from the encrypted shares. As we explain in Section 2, recovering b_i and g^{r_i} along with the proofs of share validity is sufficient for transferring the secret deposit.

In Figure 3.3, we present Protocol $\Pi_{\mathcal{C}}$ followed by the committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and executed as part of Protocols Π_{FPA} (resp. Π_{SPA}) described

in Section 3.6 (resp. Section 3.7). The interaction of the other parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ executing Protocols Π_{FPA} and Π_{SPA} with the committee $\mathcal{C}$ is described as part of Stage 1 - Setup of these protocols.

Selecting Committees:

In order to focus on the novel aspects of our constructions, we assume that the smart contract captured by $\mathcal{F}_{\mathsf{SC}}$ described in Section 3.4 is parameterized by a description of the committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and the public keys corresponding to each committee member. Notice that in practice this committee can be selected by the smart contract from the set of parties executing the underlying blockchain consensus protocol. The problem of selecting committees in a permissionless blockchain scenario has been extensively addressed in both Proof-of-Stake [126, 84, 70] and Proof-of-Work [155] settings.

3.6 First-Price Auctions

In this section, we introduce our protocol for first-price auctions (while the case of second-price auctions is addressed in Section 3.7). We consider a setting with n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where each party $\mathcal{P}_i$ has a l-bit bid $b_i = b_{i1}| \ldots |b_{il}$, where b_{ir} denotes the r-th bit of party $\mathcal{P}_i$'s bid.

Modelling Fair Auctions:

First, we introduce an ideal model for fair auctions that we will use to prove the security of our protocol. For the sake of simplicity, when discussing this model, we use $\mathsf{coins}(n)$ to indicate n currency tokens being transferred where n is represented in binary, instead of describing a full UTXO transaction. Our ideal functionality $\mathcal{F}_{FPA}$ is described in Figure 3.4. This functionality models the fact that the adversary may choose to abort but all it may learn is that it was the winner and the most significant bit where its bid differs from the second-highest bid. Regardless of adversarial actions, an auction result is always obtained and the auctioneer (*i.e.*, the party selling the asset) is always paid. The second-price case is presented in Section 3.7.

The Protocol:

In Figures 3.5, 3.6, 3.7 and 3.8, we construct a Protocol Π_{FPA} that realizes $\mathcal{F}_{FPA}$. This protocol is executed by n parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, where each party $\mathcal{P}_i$ has a l-bit bid $b_i = b_{i1}| \ldots |b_{il}$ and a deposit committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ that helps open secret deposits from corrupted parties in the Recovery Stage. The protocol consists of 4 main stages plus a recovery stage, which is only executed in case of suspected (or detected) cheating. In the first stage, every party i sends to the smart contract a secret deposit, whose structure will be explained in detail later. In the second and third stage, all parties jointly compute the maximum bid (bit-by-bit) by using an anonymous veto protocol that computes a logical OR on private inputs. To this aim, the parties start from the most significant bit

Functionality $\mathcal{F}_{FPA}$

$\mathcal{F}_{FPA}$ operates with an auctioneer $\mathcal{P}_{AUC}$, a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ who have bids $b_1, \ldots, b_n$ as input and where $b_i = b_{i1}| \ldots |b_{il}$ is the bit representation of b_i, as well as an adversary $\mathcal{S}_{FPA}$. $\mathcal{F}_{FPA}$ is parameterized by a bid bit-length l and keeps an initially empty list bids.

- **Setup (Bid Registration):** Upon receiving $(\text{BID}, sid, \mathsf{coins}(b_i + work))$ from $\mathcal{P}_i$ where $b_i \in \{0,1\}^l$ and $work$ is the amount required to compensate the cost of running the protocol for all the other parties, $\mathcal{F}_{FPA}$ appends b_i to bids.
- **First-Price Auction:** After receiving $(\text{BID}, sid, \mathsf{coins}(b_i + work))$ from all parties in $\mathcal{P}$, for $r = 1, \ldots, l$ $\mathcal{F}_{FPA}$ proceeds as follows:
  1. Select b_{wr}, *i.e.*, the r-th bit of the highest bid b_w in the list bids.
  2. Send $(\text{ROUND-WINNER}, sid, b_{wr})$ to all parties and $\mathcal{S}_{FPA}$.
  3. Check if $b_{wr} = 1$ and $b_{ir} = 0$ for $i = 1, \ldots, n \neq w$. If so, let $r_w = r$, that is the first position where b_w has a bit 1 and b_{w_2} has a bit 0, and send $(\text{LEAK-TO-WINNER}, sid, r_w)$ to $\mathcal{P}_w$.
  4. Send $(\text{ABORT?}, sid)$ to $\mathcal{S}_{FPA}$. If $\mathcal{S}_{FPA}$ answers with $(\text{ABORT}, sid, \mathcal{P}_i)$ where $\mathcal{P}_i$ is corrupted, remove b_i from bids, remove $\mathcal{P}_i$ from $\mathcal{P}$, send $(\text{ABORT}, sid, \mathcal{P}_i, \mathsf{coins}(\frac{b_i + work}{|\mathcal{P}|}))$ where $|\mathcal{P}|$ is the number of remaining parties to all other parties in $\mathcal{P}$, set again $r = 1$ and go to Step 1. If $\mathcal{S}_{FPA}$ answers with $(\text{PROCEED}, sid)$, if $r = l$ go to Payout, else increment r by 1 and go to Step 1.
- **Payout:** Send $(\text{REFUND}, sid, \mathcal{P}_w, \mathsf{coins}(b_i + work))$ to all parties $\mathcal{P}_i \neq \mathcal{P}_w$, send $(\text{REFUND}, sid, \mathsf{coins}(work))$ to $\mathcal{P}_w$, send $\mathsf{coins}(b_w)$ to $\mathcal{P}_{AUC}$, and halt.

Figure 3.4: Functionality $\mathcal{F}_{FPA}$.

position. Then, they apply the anonymous veto protocol according to their bits b_{ir}, with 0 representing a no veto and 1 representing a veto. If the outcome of the veto protocol (*i.e.*, the logical-OR of the the inputs) is 1, then each party $\mathcal{P}_i$ with input $b_{ir} = 0$ figures out that there is at least another party $\mathcal{P}_k$ whose bid b_k is higher than b_i and $\mathcal{P}_i$ discovers that she cannot win the auction. Therefore, from this point on, $\mathcal{P}_i$ stops vetoing, disregarding her actual bit b_{ir} in the next rounds. Otherwise, $\mathcal{P}_i$ is expected to keep vetoing or not according to her bit b_{ir}. Finally, in Stage 4 the winning party $\mathcal{P}_w$ executes the payment to the auctioneer (*i.e.*, the party selling the asset). Throughout all stages, the parties must provide proofs that they have correctly computed all protocol messages (using the NIZKs described in Section 3.3). If a party is identified as dishonest at any point, the Recovery Stage has to be executed.

Security Analysis:

It is clear that this protocol correctly computes the highest bid. The ideal smart contract enforces payment once a winner is determined and punishments otherwise. The security of this protocol is formally stated in the following theorem, which is proven in Section 3.6.1. A game-theoretical analysis is presented in

Section 3.9, where it is shown that the best strategy for any rational party is to follow the protocol.

Theorem 1. *Under the DDH Assumption, Protocol Π_{FPA} securely computes $\mathcal{F}_{FPA}$ in the $\mathcal{F}_{\mathsf{SC}}$-hybrid, random oracle model against a malicious static adversary $\mathcal{A}$ corrupting all but one parties $\mathcal{P}_i \in \mathcal{P}$ and $m/2-2$ parties $\mathcal{C}_i \in \mathcal{C}$.*

Protocol Π_{FPA} (Off-chain messages exchange)

Protocol Π_{FPA} is executed with n parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, where each party $\mathcal{P}_i$ has a l-bit bid $b_i = b_{i1}|\ldots|b_{il}$ and a deposit committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$. Parties $\mathcal{P}, \mathcal{C}$ interact among themselves and with a smart contract $\mathcal{F}_{\mathsf{SC}}$.

Off-chain messages exchange: To minimize the communication with the smart contract, an approach based on [18] is adopted. Let r be a generic round of the protocol, then each party $\mathcal{P}_i$ actually proceeds as follows when sending her messages:

- **Round**$_r$: each $\mathcal{P}_i$ sends $msg_{r,i}$, $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{r,i})$ to all the other parties;
- **Round**$_{r+1}$: all the other parties $\mathcal{P}_k$ for $k \in \{1, \ldots, n\} \setminus i$ sign the message received from party i and send $msg_{r,i}$, $\mathsf{Sig.Sign}_{\mathsf{sk}_k}(msg_{r,i})$ to all the other parties, allowing them to check if party i sent no conflicting messages. Then, each party repeats from the instructions described in the previous round;
- **Conflicting messages**: in case $\mathcal{P}_i$ sends conflicting messages $msg_{r,i} \neq msg'_{r,i}$ to parties $\mathcal{P}_k \neq \mathcal{P}_{k'}$, $\mathcal{P}_k$ or $\mathcal{P}_{k'}$ send to the smart contract $msg_{r,i}$, $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{r,i})$ and $msg'_{r,i}$, $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg'_{r,i})$ as a proof that $\mathcal{P}_i$ was dishonest;
- **Evidence of a message**: in case it has to be proven that a message $msg_{r,i}$ has been sent by party $\mathcal{P}_i$ in round r, the other parties send to the smart contract the signatures $\mathsf{Sig.Sign}_{\mathsf{sk}_1}(msg_{r,i}), \ldots, \mathsf{Sig.Sign}_{\mathsf{sk}_n}(msg_{r,i})$ along with the message $msg_{r,i}$.

Figure 3.5: Protocol Π_{FPA} (Off-chain messages exchange).

3.6.1 Proof of Theorem 1

In this section, we provide a full proof of security for Protocol Π_{FPA}. In order to prove Theorem 1, we first the following auxiliary Lemmas:

Lemma 1. *Under the DDH Assumption, the Pedersen commitment scheme [156] is computationally binding and unconditionally hiding:*

- *Computationally binding: under the DL assumption, for any PPT algorithm the probability $\epsilon(q)$ of finding $s_1, t_1, s_2, t_2 \in \mathbb{Z}_q$ such that $s_1 \neq s_2$ and $\mathsf{com}(s_1, t_2) = \mathsf{com}(s_2, t_2)$ is $\mathsf{negl}(q)$.*
- *Unconditionally hiding: for any $s_1, s_2 \in \mathbb{Z}_q$ and $t_1, t_2 \xleftarrow{\$} \mathbb{Z}_q$, it holds that $|\Pr[\mathcal{D}(\mathsf{com}(s_1, t_1)) = 1] - \Pr[\mathcal{D}(\mathsf{com}(s_2, t_2)) = 1]| = \mathsf{negl}(q)$ for any*

Protocol Π_{FPA} (Stage 1)

Stage 1 - Setup: Deposit committee parties $\mathcal{C}_k \in \mathcal{C}$ first execute the Setup Verification step of $\Pi_{\mathcal{C}}$ from Figure 3.3. All parties $\mathcal{P}_i$ proceed as follows:

1. $\mathcal{P}_i$ sends a secret deposit containing their bid b_i, change *change* and a fee *work* to the smart contract through a confidential transaction (as described in Section 2.6.2). Let $Addr_i$ be the address associated to party i and $Addr_s$ be the address associated to the smart contract, $\mathcal{P}_i$ proceeds as follows:
   (a) $\mathcal{P}_i$ sends (PARAM, sid) to $\mathcal{F}_{\mathsf{SC}}$, receiving $(\text{PARAM}, sid, g, h, \mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m})$.
   (b) $\mathcal{P}_i$ computes the bit commitments as $c_{ir} = g^{b_{ir}} h^{r_{ir}}$, with $r_{ir} \xleftarrow{\$} \mathbb{Z}_q$, to each bit b_{ir} of b_i, and the bid commitment as $c_i = \prod_{r=1}^{l} c_{ir}^{2^{l-r}} = g^{b_i} h^{\sum_{r=1}^{l} 2^{l-r} r_{ir}}$. Let r_{b_i} be equal to $\sum_{r=1}^{l} 2^{l-r} r_{ir}$. Then, c_i can be rewritten as $c_i = g^{b_i} h^{r_{b_i}} = \mathsf{com}(b_i, r_{b_i})$.
   (c) Define sets $\mathsf{In} = \{(\mathtt{id}_i, \mathtt{in}_i)\}$ and $\mathsf{Out} = \{(c_i, Addr_s), (work, Addr_s), (\mathsf{com}(change_i, r_{change_i}), Addr_i)\}$, where $c_i = \mathsf{com}(b_i, r_{b_i})$ is the commitment to the bid b_i previously computed at Step 1, *work* is the amount required to compensate the cost of running the protocol for all the other parties in $\mathcal{P}$ and in $\mathcal{C}$, $change = \mathtt{in}_i - b_i - work$ and $r_{change} \xleftarrow{\$} \mathbb{Z}_q$. Note that, in this case case, $\mathtt{in}_i$ and *work* are public, while b_i and *change* are private.
   (d) Compute $r_{out} = r_{b_i} + r_{change_i}$, so as to allow the other parties later to verify that the sum of the inputs is equal to the sum of the outputs, *i.e.*, $c_i \cdot \mathsf{com}(change, r_{change}) \stackrel{?}{=} \mathsf{com}(\mathtt{in}_i - work, r_{out})$.
   (e) Compute proofs $(\pi_{b_i}, \pi_{change})$ showing that $b_i, change \in [0, 2^l - 1]$, set $\mathsf{tx}_i = (\mathsf{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig}, r_{b_i} + r_{change_i}, \pi)$.
   (f) Compute the shares $(\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}, LDEI_i)$ of g^{b_i} and $h^{r_{b_i}}$ using the distribution procedure from π_{PVSS} with $\mathsf{pk}_{\mathcal{C}_1}, \ldots, \mathsf{pk}_{\mathcal{C}_m}$ received in step (a).
   (g) Compute $CC_i \leftarrow CC((\mathsf{pk}_{\mathcal{C}_j})_{j\in[m]}, (j)_{j\in[m]}, g, h, c_i, (\hat{\sigma}_j)_{j\in[m]})$ to prove consistency among the shares $(\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im})$ and the commitment terms g^{b_i} and $h^{r_{b_i}}$ from $c_i = g^{b_i} h^{r_{b_i}}$.
   (h) Send $(\text{SETUP}, sid, \mathcal{P}_i, \mathsf{tx}_i, \mathsf{pk}_i, (\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i)$ to each $\mathcal{C}_j \in \mathcal{C}$.
   (i) Upon receiving $(\text{SETUP-VERIFICATION}, sid, Sig_{\mathcal{C}_j,i})$ from all $\mathcal{C}_j \in \mathcal{C}$, compute $SH1_i = \mathcal{H}(\mathsf{tx}_i, \mathsf{pk}_i)$ and $SH2_i = \mathcal{H}((\hat{\sigma}_{i1}, \ldots, \hat{\sigma}_{im}), LDEI_i, CC_i)$ and send $(\text{SETUP}, sid, \mathcal{P}_i, \mathsf{tx}_i, \mathsf{pk}_i, SH1_i, SH2_i, Sig_{\mathcal{C}_1,i}, \ldots, Sig_{\mathcal{C}_m,i}))$ to $\mathcal{F}_{\mathsf{SC}}$. If a party $\mathcal{C}_a \in \mathcal{C}$ does not send this message, proceed to the Recovery Stage.
2. $\mathcal{P}_i$ samples $x_{ir} \xleftarrow{\$} \mathbb{Z}_q$ and computes $X_{ir} = g^{x_{ir}}$ for $r = 1, \ldots, l$, sending $c_{i1}, \cdots, c_{il}$, $X_{i1}, \cdots, X_{il}$ to all other parties.
3. Upon receiving all messages $c_{j1}, \cdots, c_{jl}$, $X_{j1}, \cdots, X_{jl}$ from other parties $\mathcal{P}_j$, $\mathcal{P}_i$ computes $Y_{jk} = \prod_{m=1}^{j-1} X_{mk} / \prod_{m=j+1}^{n} X_{mk}$ for $j = 1, \ldots, n$,$k = 1, \ldots, l$, and verifies for each other party $\mathcal{P}_j$ that $c_j = \prod_{k=1}^{l} c_{jk}^{2^{l-k}}$ for $j \in \{1, \ldots, n\} \setminus i$. If this verification fails or a message is not received, proceed to the Recovery Stage.

Figure 3.6: Protocol Π_{FPA} (Stage 1).

Protocol Π_{FPA} (Stages 2 and 3)

Stage 2 - Before First Veto: All parties $\mathcal{P}_i$, starting from the most significant bit b_{i1} and moving bit-by-bit to the least significant bit b_{il} of their bid $b_i = b_{i1}|\dots|b_{il}$, run in each round r the anonymous veto protocol until the outcome is a veto (*i.e.*, $V_r \neq 1$) for the first time. Therefore each party $\mathcal{P}_i$ proceeds as follows:

1. Compute v_{ir} as follows:

$$v_{ir} = \begin{cases} Y_{ir}^{x_{ir}}, & \text{if } b_{ir} = 0 \\ \bar{r}_{ir} \xleftarrow{\$} \mathbb{Z}_q, g^{\bar{r}_{ir}}, & \text{if } b_{ir} = 1 \end{cases}$$

and generate NIZK proving that v_{ir} has been correctly computed $BV_{ir} \leftarrow BV\{b_{ir}, r_{ir}, x_{ir}, \bar{r}_{ir} \mid (\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}) \vee (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge v_{ir} = g^{\bar{r}_{ir}})\}$, sending a message (v_{ir}, BV_{ir}) to all parties.

2. Upon receiving all messages (v_{kr}, BV_{kr}) from other parties $\mathcal{P}_k$, $\mathcal{P}_i$ checks the proofs BV_{kr} for $k \in \{1, \dots, n\} \setminus i$ and, if all checks pass, computes $V_r = \prod_{k=1}^{n} v_{kr}$ and then goes to Stage 3 if $V_r \neq 1$ (at least one veto), otherwise follows the steps in Stage 2 again until the round $r = l$. Note that, unless all the bids are equal to 0, at some point the condition $V_r \neq 1$ is satisfied. If a message is not received from party $\mathcal{P}_k$ or if BV_{kr} is invalid, proceed to the Recovery Stage.

Stage 3 - After First Veto: Let $\widehat{r}$ denote the last round at which there was a veto (*i.e.*, $V_{\widehat{r}} \neq 1$). All parties $\mathcal{P}_i$, starting from $b_{i\widehat{r}+1}$ and moving bit-by-bit to the least significant bit b_{il} of their bid $b_i = b_{i1}|\dots|b_{il}$, run in each round $r > \widehat{r}$ the anonymous veto protocol taking into account both the input bit b_{ir} and the declared input bit d_{ir}, defined as the value that satisfies the logical condition $(b_{ir} = 0 \wedge d_{ir} = 0) \vee (b_{ir} = 1 \wedge d_{i\widehat{r}} = 1 \wedge d_{ir} = 1) \vee (b_{ir} = 1 \wedge d_{i\widehat{r}} = 0 \wedge d_{ir} = 0)$, *i.e.*, each party $\mathcal{P}_i$ vetoes at round r iff she also vetoed at round $\widehat{r}$ (*i.e.*, $d_{i\widehat{r}} = 1$), and her current input bit $b_{ir} = 1$. Therefore, each $\mathcal{P}_i$ proceeds as follows:

1. Compute v_{ir} as follows:

$$v_{ir} = \begin{cases} Y_{ir}^{x_{ir}}, & \text{if } b_{ir} = 0 \\ \bar{r}_{ir} \xleftarrow{\$} \mathbb{Z}_q, g^{\bar{r}_{ir}}, & \text{if } d_{i\widehat{r}} = 1 \wedge b_{ir} = 1 \\ Y_{ir}^{x_{ir}}, & \text{if } d_{i\widehat{r}} = 0 \wedge b_{ir} = 1 \end{cases}$$

and generate NIZK proving that v_{ir} has been correctly computed $AV_{ir} \leftarrow AV\{b_{ir}, r_{ir}, x_{ir}, \bar{r}_{i\widehat{r}}, \bar{r}_{ir}, x_{i\widehat{r}} \mid (\frac{c_{ir}}{g^{b_{ir}}} = c_{ir} = h^{r_{ir}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}}) \vee (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge d_{i\widehat{r}} = g^{\bar{r}_{i\widehat{r}}} \wedge v_{ir} = g^{\bar{r}_{ir}}) \vee (\frac{c_{ir}}{g^{b_{ir}}} = \frac{c_{ir}}{g} = h^{r_{ir}} \wedge \ d_{i\widehat{r}} = Y_{i\widehat{r}}^{x_{i\widehat{r}}} \wedge X_{i\widehat{r}} = g^{x_{i\widehat{r}}} \wedge v_{ir} = Y_{ir}^{x_{ir}} \wedge X_{ir} = g^{x_{ir}})\}$, sending a message (v_{ir}, AV_{ir}) to all parties.

2. Upon receiving all messages (v_{kr}, AV_{kr}) from other parties $\mathcal{P}_k$, $\mathcal{P}_i$ checks the proofs AV_{kr} for $k \in \{1, \dots, n\} \setminus i$ and, if all checks pass, computes $V_r = \prod_{k=1}^{n} v_{kr}$, following the steps in Stage 3 again until round $r = l$. If a message is not received from party $\mathcal{P}_k$ or if AV_{kr} is invalid, proceed to the Recovery Stage.

Figure 3.7: Protocol Π_{FPA} (Stages 2 and 3).

Protocol Π_{FPA} (Stages 4 and Recovery)

Stage 4 - Output: At this point, each party $\mathcal{P}_i$ knows the value of V_r for each round $r = 1, \cdots, l$ and the protocol proceeds as follows:

1. $\mathcal{P}_i$ computes the winning bid as $b_w = b_{w1}|\cdots|b_{wl}$, such that $b_{wr} = 1$ if $V_r \neq 1$ and $b_{wr} = 0$ if $V_r = 1$, and sends b_w to all other parties (causing all parties $\mathcal{P}_k$ to sign b_w and send $\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)$ to each other). We denote by $\mathcal{P}_w$ the winning party (*i.e.*, the party whose bid is b_w).
2. $\mathcal{P}_w$ opens the commitment to her bid $\mathsf{com}(b_w, r_{b_w})$ towards the smart contract by sending $(\text{OUTPUT}, sid, \mathcal{P}_w, b_w, r_{b_w}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]})$ to $\mathcal{F}_{\mathsf{SC}}$.
3. If $\mathcal{P}_w$ does not open her commitment or if multiple parties open their commitments, $\mathcal{P}_i$ proceeds to the Recovery Stage.
4. Finally, all parties who honestly completed the execution of the protocol receive a refund of their deposit from the smart contract, apart from the winning party, who only receives a refund equivalent to the *work* funds.

Recovery Stage: Parties $\mathcal{C}_i \in \mathcal{C}$ listen to $\mathcal{F}_{\mathsf{SC}}$ and execute the Share Decryption step of $\Pi_{\mathcal{C}}$ from Figure 3.3 if requested. In case a party $\mathcal{P}_i \in \mathcal{P}$ is suspected of cheating, the Recovery stage is executed as follows to identify the cheater depending on the exact suspected cheating:

- **Missing message or signatures**: a message msg_{ri} or a signature $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{r-1,i'})$, on a message $msg_{r-1,i'}$ by $\mathcal{P}'_i$, expected to be sent in round r by $\mathcal{P}_i$ is not received by $\mathcal{P}_k$. Then, $\mathcal{P}_k$ sends to $\mathcal{F}_{\mathsf{SC}}$ the message $(\text{RECOVERY-MISSING}, sid, msg, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(msg)\}_{k\in[n]})$, where msg is the last message signed by all parties and waits for $\mathcal{F}_{\mathsf{SC}}$ to request the missing message. In that way, $\mathcal{P}_i$ is expected to send msg_{ri} or $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{r-1,i'})$ to $\mathcal{F}_{\mathsf{SC}}$. If no action is taken, $\mathcal{P}_i$ is identified as a cheater.
- **Conflicting messages or Invalid message**: In round r, $\mathcal{P}_i$ sends conflicting messages msg_{ri}, $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri})$ and msg'_{ri}, $sig_{\mathsf{sk}_i}(msg'_{ri})$ to different parties $\mathcal{P}_k$ and $\mathcal{P}'_k$. In this case, $\mathcal{P}_k$ and $\mathcal{P}'_k$ set the conflicting messages as a proof of cheating $\pi_c = (msg_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri}), msg'_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg'_{ri}))$. Otherwise, $\mathcal{P}_i$ sends an invalid message $msg_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri})$ to $\mathcal{P}_k$ (*i.e.*, the message does not follow the structure described in the protocol for messages in round r), $\mathcal{P}_k$ uses this message as a proof of cheating $\pi_c = (msg_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri}))$. $\mathcal{P}_k$ sends $(\text{RECOVERY-CHEAT}, sid, \mathcal{P}_i, \pi_c)$ to the smart contract and $\mathcal{P}_i$ is identified as a cheater.

Every party $\mathcal{P}_i$ identified as a cheater loses her whole deposit $(b_i + work)$, which is distributed to the other parties by $\mathcal{F}_{\mathsf{SC}}$, and the protocol continues as follows:

- **Re-execution (unknown b_w)**: in case b_w has not been computed, the protocol is re-executed from Stage 2 excluding the parties identified as cheaters.
- **Complete payment (known b_w but unknown $\mathcal{P}_w$)**: in case b_w has been computed but $\mathcal{P}_w$ does not send $(\text{OUTPUT}, sid, \mathcal{P}_w, b_w, r_{b_w}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]})$ to $\mathcal{F}_{\mathsf{SC}}$, all $\mathcal{P}_i \in \mathcal{P}$ compute a NIZK $NW_i \leftarrow NW\{x_{i1}, \ldots, x_{il} \mid (V_1 = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}}) \vee \ldots \vee (V_l = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}})\}$ showing that they are not the winner. Then they send to $\mathcal{F}_{\mathsf{SC}}$ $(\text{RECOVERY-PAYMENT}, sid, NW_i)$. The winner $\mathcal{P}_w$ (in case it is identified) or all parties $\mathcal{P}_i$ who do not act (in case $\mathcal{P}_w$ is not identified) are identified as dishonest and lose their deposits, which are distributed among the honest parties.

Figure 3.8: Protocol Π_{FPA} (Stages 4 and Recovery).

distiguisher $\mathcal{D}$, i.e., $\{\mathsf{com}(s_1,t_2)\}_{s_1\in\mathbb{Z}_q,t_1\xleftarrow{\$}\mathbb{Z}_q}$ and $\{\mathsf{com}(s_1,t_2)\}_{s_2\in\mathbb{Z}_q,t_2\xleftarrow{\$}\mathbb{Z}_q}$ are statistically indistiguishable.

Proof. It is proven in [156] that the Pedersen commitment scheme is computationally binding and unconditionally hiding under the assumption that the Discrete Logarithm problem is hard, which is implied by the DDH assumption. □

Lemma 2. *Under the DDH Assumption and in the random oracle model, there exists a EUF-CMA [109] secure digital signature scheme.*

Proof. There exist a number of digital signature schemes whose security is implied by the DDH assumption in the random oracle model, *e.g.*, [159] and [34]. □

Lemma 3 (Theorem 5 from [48])**.** *Under the DDH Assumption and in the random oracle model, the range proofs computed in Stage 1 of Π_{FPA} guarantee the zero-knowledge, the proof of knowledge and the soundness properties.*

Lemma 4 (From [58])**.** *Under the DDH Assumption and in the random oracle model, the NIZKs BV , AV, NW and CC computed respectively in Stage 2, Stage 3 and the Recovery Stage of Π_{FPA} have the zero-knowledge, the proof of knowledge and the soundness properties.*

Lemma 5 (Lemma 2 from [13])**.** *Under the DDH Assumption, given $X_{ir} = g^{x_{ir}}$ with $x_{ir} \xleftarrow{\$} \mathbb{Z}_q$ and $i \in [1,n]$, $Y_{ir} = \prod_{k=1}^{i-1} g^{x_{kr}} / \prod_{k=i+1}^{n} g^{x_{kr}} = g^{(\sum_{k=1}^{i-1} x_{kr} - \sum_{k=i+1}^{n} x_{kr})}$ and $y_{ir} = \sum_{k=1}^{i-1} x_{kr} - \sum_{k=i+1}^{n} x_{kr}$ with $i \in [1,n]$, $i \in [1,n]$, $i', i'' \in [1,n]$ such that $i' \neq i''$, $g^{\bar{r}_{ir}}$ with $\bar{r}_{ir} \xleftarrow{\$} \mathbb{Z}_q$ and $i \in [1,n] \setminus \{i', i''\}$, $g^{\bar{r}_{i'r}}$, $g^{\bar{r}_{i''r}}$, $\Phi \subseteq \{x_{ir} : i \in [1,n] \setminus \{i', i''\}\}$ and a challenge $\Omega \in \{A, B\}$, it is computationally hard to find if $\Omega = A$ or $\Omega = B$, where:*

$$A = (g, \Phi, g^{x_{1r}z_{1r}}, g^{x_{2r}z_{2r}}, \ldots, g^{x_{i'-1r}z_{i'-1r}}, g^{x_{i'r}\bar{r}_{i'r}}, g^{x_{i'+1r}z_{i'+1r}}, \ldots, g^{x_{i''r}y_{i''r}}, \ldots, g^{x_{nr}z_{nr}})$$

$$B = (g, \Phi, g^{x_{1r}z_{1r}}, g^{x_{2r}z_{2r}}, \ldots, g^{x_{i'-1r}z_{i'-1r}}, g^{x_{i'r}\bar{r}_{i'r}}, g^{x_{i'+1r}z_{i'+1r}}, \ldots, g^{x_{i''r}\bar{r}_{i''r}}, \ldots, g^{x_{nr}z_{nr}})$$

where z_{ir} is either y_{ir} (note that when $z_{ir} = y_{ir}$ the value $g^{x_{ir}z_{ir}}$ is equal to the message $v_{ir} = Y_{ir}^{x_{ir}} = g^{x_{ir}y_{ir}}$ of Π_{FPA} representing a no veto) or $\bar{r}_{ir}$ (note that when $z_{ir} = \bar{r}_{ir}$, where $\bar{r}_{ir} \xleftarrow{\$} \mathbb{Z}_q$, the value $g^{x_{ir}z_{ir}}$ is indistinguishable from the message $v_{ir} = g^{\bar{r}_{ir}}$ of Π_{FPA} representing a veto) for $i \in [1,n] \setminus \{i', i''\}$, and Φ is chosen by an adversary $\mathcal{A}$. Intuitively, it is not possible to distinguish two executions in which there is at least one veto but the number of parties vetoing is different and it is not possible to learn if a party vetoed or not by checking v_{ir}.

Lemma 6 (Lemma 3 from [13])**.** *Under the DDH Assumption, let $\mathcal{H}$ be a set of honest parties and $\mathcal{C}$ be a set of parties corrupted by an adversary $\mathcal{A}$. For each $\mathcal{P}_h \in \mathcal{H}$, let v_{hr} be her message in Stages 2 or 3 during a round r of Π_{FPA}, corresponding to an input bit b_{hr}. Then, $\mathcal{A}$ learns no more than $\bigvee_{\mathcal{P}_h\in\mathcal{H}} b_{hr}$. In particular, in case $\bigvee_{\mathcal{P}_h\in\mathcal{H}} b_{hr} = 0$, $\mathcal{A}$ learns that $b_{hr} = 0$ for each $\mathcal{P}_h \in \mathcal{H}$. On the other hand, in case $\bigvee_{\mathcal{P}_h\in\mathcal{H}} b_{hr} = 1$:*

- *$\mathcal{A}$ is not able to distinguish two executions in which the number of honest parties $\mathcal{P}_h \in \mathcal{H}$ with $b_{hr} = 1$ is different.*
- *Let $\mathcal{P}_{h_1}, \mathcal{P}_{h_2} \in \mathcal{H}$ be honest parties with input bits $b_{h_1 r}, b_{h_2 r}$ such that $b_{h_1 r} \neq b_{h_2 r}$ and messages $v_{h_1 r}, v_{h_2 r}$ respectively. Then, $\mathcal{A}$ is not able to distinguish $v_{h_1 r}$ and $v_{h_2 r}$.*

Based on the above Lemmas, we prove Theorem 1, which we reproduce below for the sake of clarity.

Theorem 1. *Under the DDH Assumption, Protocol Π_{FPA} securely computes $\mathcal{F}_{FPA}$ in the $\mathcal{F}_{\mathsf{SC}}$-hybrid, random oracle model against a malicious static adversary $\mathcal{A}$ corrupting all but one parties $\mathcal{P}_i \in \mathcal{P}$ and $m/2 - 2$ parties $\mathcal{C}_i \in \mathcal{C}$.*

Proof. In order to prove this theorem, we construct a simulator $\mathcal{S}_{FPA}$ (Figures 3.9, 3.10, 3.11 and 3.12) that performs an ideal execution with $\mathcal{F}_{FPA}$ and interacts with an internal copy of the adversary $\mathcal{A}$, simulates honest parties, $\mathcal{F}_{\mathsf{SC}}$ and the random oracle in an execution of Protocol Π_{FPA} with $\mathcal{A}$ in such a way that this execution is indistinguishable from an execution between $\mathcal{A}$ and an honest party in the real world. Throughout this execution, $\mathcal{S}_{FPA}$ perfectly emulates $\mathcal{F}_{\mathsf{SC}}$ and the random oracle unless stated otherwise. In order to show that an ideal execution with $\mathcal{S}_{FPA}$ and $\mathcal{F}_{FPA}$ is indistinguishable from a real execution of Π_{FPA} with $\mathcal{A}$ and honest parties, we argue that the view of $\mathcal{A}$ in the real world and of $\mathcal{S}_{FPA}$'s internal copy of $\mathcal{A}$ is indistinguishable. In particular:

- **Stage 1 - Setup:** $\mathcal{S}_{FPA}$ simulates each honest party $\mathcal{P}_h \in \mathcal{H}$ using a dummy input bid $b_h = b_{h1}|\ldots|b_{hl}$ with $b_{hr} \xleftarrow{\$} \{0,1\}$ for $r = 1, \ldots, l$ and computes the bit commitments $c_{hr} = g^{b_{hr}} h^{r_{hr}}$ for $r = 1, \ldots, l$, the bid commitment $c_h = \prod_{r=1}^{l} c_{hr}^{2^{l-r}} = g^{b_h} h^{\sum_{r=1}^{l} 2^{l-r} r_{hr}}$, the range proofs $(\pi_{b_h}, \pi_{change})$, shares $(\hat{\sigma}_{h1}, \ldots, \hat{\sigma}_{hm}, LDEI_h)$ of $g^{b'_h}$ and $h^{r_{b'_h}}$ and the proof of consistency CC_h.

  However, by Lemma 1, due to the hiding property of the commitments, c_{hr} for $r = 1, \ldots, l$ and c_h are indistinguishable from the commitments of the corresponding parties in the real world. Moreover, by Lemma 3, due to the zero knowledge property of NIZKs, $(\pi_{b_h}, \pi_{change})$ are indistinguishable from the range proofs of the corresponding parties in the real world.

  Similarly, by Lemma 4, due to the zero knowledge property of NIZKs, CC_h is indistinguishable from the NIZK of the corresponding party in the real world.

  On the other hand, by Lemma 1, due to the binding property of the commitment, the bids b_c of each corrupted party $\mathcal{P}_a \in \mathcal{C}$, extracted by $\mathcal{S}_{FPA}$ using the NIZKs knowledge extractor (given the proof of knowledge property of NIZKs by Lemma 3), from the range proofs $(\pi_{b_a}, \pi_{change})$, cannot be changed later. Then, by Lemma 3, due to the soundness property of the NIZKs, it is computationally hard for the adversary $\mathcal{A}$ controlling each $\mathcal{P}_a \in \mathcal{C}$ to compute the range proofs while the bids are not in the

Simulator $\mathcal{S}_{FPA}$ (Stage 1)

Let $\mathcal{H}$ be the set of simulated honest parties, $\mathcal{C}$ be the set of parties corrupted by the adversary $\mathcal{A}$, $\mathcal{C}_{\mathcal{H}}$ be the set of simulated members of the committee and $\mathcal{C}_{\mathcal{C}}$ be the set of members of the committee corrupted by the adversary $\mathcal{A}$.

Stage 1 - Setup.

- **Generating Parameters with Trapdoor:** $\mathcal{S}_{FPA}$ samples $g \xleftarrow{\$} \mathbb{G}$, $t \xleftarrow{\$} \mathbb{Z}_q$ and computes $h = g^t$. Trapdoor t will allow $\mathcal{S}_{FPA}$ to equivocate commitments later. When $\mathcal{A}$ queries $\mathcal{F}_{\mathsf{SC}}$ with (PARAM, sid), $\mathcal{S}_{FPA}$ returns these values (PARAM, sid, g, h).
- **Simulating Honest Parties:** $\mathcal{S}_{FPA}$ simulates each honest party $\mathcal{P}_h \in \mathcal{H}$ using dummy input bids $b'_h = b'_{h1}|\dots|b'_{hl}$ such that $b'_{hr} \xleftarrow{\$} \{0,1\}$ for $r = 1, \dots, l$. $\mathcal{S}_{FPA}$ follows the steps of an honest party with input b'_h in Π_{FPA} to do the following:
  1. Generate pk_h, transaction tx_h, shares $(\hat{\sigma}_{h1}, \dots, \hat{\sigma}_{hm}, LDEI_h)$ of $g^{b'_h}$ and $h^{r_{b'_h}}$, proof of consistency $CC_h \leftarrow CC((\mathsf{pk}_{\mathcal{C}_j})_{j\in[m]}, (j)_{j\in[m]}, g, h, c_h, (\hat{\sigma}_j)_{j\in[m]})$, then send (SETUP, $sid, \mathcal{P}_h, \mathsf{tx}_h, \mathsf{pk}_h, (\hat{\sigma}_{h1}, \dots, \hat{\sigma}_{hm}), LDEI_h, CC_h$) to each $\mathcal{C}_j \in \mathcal{C}_{\mathcal{C}}$.
  2. Upon receiving (SETUP-VERIFICATION, $sid, Sig_{\mathcal{C}_j,h}$) from all $\mathcal{C}_j \in \mathcal{C}_{\mathcal{C}}$, compute $SH1_h = \mathcal{H}(\mathsf{tx}_h, \mathsf{pk}_h)$ and $SH2_h = \mathcal{H}((\hat{\sigma}_{h1}, \dots, \hat{\sigma}_{hm}), LDEI_h, CC_h)$ and send (SETUP, $sid, \mathcal{P}_i, \mathsf{tx}_h, \mathsf{pk}_h, SH1_h, SH2_h, Sig_{\mathcal{C}_1,h}, \dots, Sig_{\mathcal{C}_m,h}$)) to $\mathcal{F}_{\mathsf{SC}}$.
  3. Generate values $c_{h1}, \cdots, c_{hl}, X_{h1}, \cdots, X_{hl}$ (sample $x_{hr} \xleftarrow{\$} \mathbb{Z}_q$, then $X_{hr} = g^{x_{hr}}$) and send them to all other parties.
  4. Upon receiving $X_{c1}, \cdots, X_{cl}$ from $\mathcal{A}$ for all $\mathcal{P}_c \in \mathcal{C}$, computes Y_{hk} for each $\mathcal{P}_h \in \mathcal{H}$ and $k = 1, \dots, l$.
- **Handling messages from $\mathcal{A}$:**

1. Upon receiving (SETUP, $sid, \mathcal{P}_a, \mathsf{tx}_a, \mathsf{pk}_a, (\hat{\sigma}_{a1}, \dots, \hat{\sigma}_{am}), LDEI_a, CC_a$) from $\mathcal{A}$ trough $\mathcal{P}_a \in \mathcal{C}$ to $\mathcal{C}_j \in \mathcal{C}_{\mathcal{H}}$, $\mathcal{S}_{FPA}$ follows the steps of $\Pi_{\mathcal{C}}$ to verify $\mathsf{tx}_a, (\hat{\sigma}_{a1}, \dots, \hat{\sigma}_{am}), LDEI_a, CC_a$. If all the checks pass, computes the hashes $SH1_a = \mathcal{H}(\mathsf{tx}_a, \mathsf{pk}_a)$ and $SH2_a = \mathcal{H}((\hat{\sigma}_{a1}, \dots, \hat{\sigma}_{am}), LDEI_a, CC_a)$ and the signature $Sig_{\mathcal{C}_j,a} = \mathsf{Sig.Sign}_{\mathsf{sk}'_{\mathcal{C}_j}}(SH1_a|SH2_a)$, then sends (SETUP-VERIFICATION, $sid, Sig_{\mathcal{C}_j,a}$) to $\mathcal{P}_a$.
2. Upon receiving the message (SETUP, $sid, \mathcal{P}_a, \mathsf{tx}_a, \mathsf{pk}_a, SH1_a|SH2_a$, $Sig_{\mathcal{C}_1,a}, \dots, Sig_{\mathcal{C}_m,a}$) from $\mathcal{A}$ trough $\mathcal{P}_a \in \mathcal{C}$ to $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{S}_{FPA}$ follows the steps of $\mathcal{F}_{\mathsf{SC}}$ to verify that $SH1_a = \mathcal{H}(\mathsf{tx}_a, \mathsf{pk}_a)$ and $Sig_{\mathcal{C}_j,a}$ for each $\mathcal{C}_j \in \mathcal{C}_{\mathcal{H}} \cup \mathcal{C}_{\mathcal{C}}$, continuing the execution if the checks pass and simulating the recovery procedure otherwise.
3. If the checks pass, $\mathcal{S}_{FPA}$ uses the knowledge extractor for the NIZKs in order to extract the bids b_a for each party $\mathcal{P}_a \in \mathcal{C}$ from their range proofs $(\pi_{b_a}, \pi_{change})$ included in the transactions tx_a. Then, $\mathcal{S}_{FPA}$ sends (BID, $sid, \mathsf{coins}(b_a + work)$) for each extracted bid b_a to $\mathcal{F}_{FPA}$ on behalf of $\mathcal{P}_a$.

Figure 3.9: Simulator $\mathcal{S}_{FPA}$ (Stage 1).

expected range. Moreover, by Lemma 4, due to the soundness property of the NIZKs, it is ensured that at every round the inputs of all $\mathcal{P}_a \in \mathcal{C}$ are computed according to the initial bid b_c and the protocol rules.

Moreover, by Proposition 1, $(\hat{\sigma}_{h1}, \dots, \hat{\sigma}_{hm}, LDEI_h)$ does not leak any in-

Simulator $\mathcal{S}_{FPA}$ (Stage 1 - Continuation)

Simulating an honest party vetoing or not vetoing. considering $\mathcal{P}_h \in \mathcal{H}$, in a round r during Stage 2 or Stage 3 requires $\mathcal{S}_{FPA}$ to compute v_{hr} as follows:

$$v_{hr} = \begin{cases} Y_{hr}^{x_{hr}}, & \text{if } \mathcal{S}_{FPA} \text{ simulates a no veto} \\ \bar{r}_{hr} \xleftarrow{\$} \mathbb{Z}_q, g^{\bar{r}_{hr}}, & \text{if } \mathcal{S}_{FPA} \text{ simulates a veto} \end{cases}$$

and generate NIZKs showing that v_{hr} has been computed according to both the first r bits of the dummy input bid b'_h (*i.e.* $b'_{h1}, \ldots, b'_{hr}$) and the outputs of the protocol in the previous rounds (*i.e.* $V_1, \ldots, V_{r-1}$) by using the NIZKs simulators. Whether a simulated honest party vetoes or not is decided by $\mathcal{S}_{FPA}$ according to rules, defined in Stage 2 and Stage 3 of $\mathcal{S}_{FPA}$, meaning that the behaviour of simulated honest parties is completely independent of the dummy input bid b'_h. In Stage 2, $\mathcal{S}_{FPA}$ uses the simulator of NIZK $BV\{b'_{hr}, r_{hr}, x_{hr}, \bar{r}_{hr} \mid (\frac{c_{hr}}{g^{b'_{hr}}} = c_{hr} = h^{r_{hr}} \wedge v_{hr} = Y_{hr}^{x_{hr}} \wedge X_{hr} = g^{x_{hr}}) \vee (\frac{c_{hr}}{g^{b'_{hr}}} = \frac{c_{hr}}{g} = h^{r_{hr}} \wedge v_{hr} = g^{\bar{r}_{hr}})\}$ to generate a proof BV_{hr} and sends (v_{hr}, BV_{hr}) to all parties. In Stage 3, uses the generator of NIZK $AV\{b'_{hr}, r_{hr}, x_{hr}, \bar{r}_{h\widehat{r}}, \bar{r}_{hr}, x_{h\widehat{r}} \mid (\frac{c_{hr}}{g^{b'_{hr}}} = c_{hr} = h^{r_{hr}} \wedge v_{hr} = Y_{hr}^{x_{hr}} \wedge X_{hr} = g^{x_{hr}}) \vee (\frac{c_{hr}}{g^{b'_{hr}}} = \frac{c_{hr}}{g} = h^{r_{hr}} \wedge d_{h\widehat{r}} = g^{\bar{r}_{h\widehat{r}}} \wedge v_{hr} = g^{\bar{r}_{hr}}) \vee (\frac{c_{hr}}{g^{b'_{hr}}} = \frac{c_{hr}}{g} = h^{r_{hr}} \wedge d_{h\widehat{r}} = Y_{h\widehat{r}}^{x_{h\widehat{r}}} \wedge X_{h\widehat{r}} = g^{x_{h\widehat{r}}} \wedge v_{hr} = Y_{hr}^{x_{hr}} \wedge X_{hr} = g^{x_{hr}})\}$ to generate a proof AV_{hr} and sends (v_{hr}, AV_{hr}) to all parties.

Figure 3.10: Simulator $\mathcal{S}_{FPA}$ (Stage 1 - Continuation).

formation about the bids and it is guaranteed that the shares distribution is valid.

Finally, by adopting a EUF-CMA secure digital signature, whose existence is given by Lemma 2, in each round r no $\mathcal{P}_c \in \mathcal{C}$ has the chance to forge a signature $\mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{r,c})$ for a certain message $msg_{r,c}$ pretending to be an honest party $\mathcal{P}_i$ in the real world, *i.e.*, no transaction tx_i nor a proof of cheating $\pi_c = (msg_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri}), msg'_{ri})$, indicating that $\mathcal{P}_i$ sent conflicting messages, or $\pi_c = (msg_{ri}, \mathsf{Sig.Sign}_{\mathsf{sk}_i}(msg_{ri})$, indicating that $\mathcal{P}_i$ sent an invalid message, can be forged. Additionally, $\mathcal{A}$ cannot deny having sent a message that it has signed in the past. Hence, proofs of cheating cannot be repudiated.

- **Stage 2 - Before First Veto and Stage 3 - After First Veto:** $\mathcal{S}_{FPA}$ simulates each honest party $\mathcal{P}_h \in \mathcal{H}$ vetoing or not vetoing in each round r, by using the NIZKs simulator, in a way that is decided arbitrarily by $\mathcal{S}_{FPA}$ and that is completely independent of the dummy input bid b_h.

  In particular, $\mathcal{S}_{FPA}$ simulates each honest party $\mathcal{P}_h \in \mathcal{H}$ vetoing or not vetoing coherently with the extracted input bids b_c from each party $\mathcal{P}_a \in \mathcal{C}$ corrupted by the adversary $\mathcal{A}$ and the bit b_{wr} learnt from $\mathcal{F}_{FPA}$ in each round r, *i.e.*, in case a corrupted party is known to be the winner, $\mathcal{S}_{FPA}$ simulates each honest party $\mathcal{P}_h \in \mathcal{H}$ not vetoing until $r = l$. On the other hand, in case an honest party is known to be the winner, $\mathcal{S}_{FPA}$

Simulator $\mathcal{S}_{FPA}$ (Stages 2 and 3)

- **Stage 2 - Before First Veto.** Let r_v be the first round in which a veto occurs, *i.e.*, $V_{r_v} \neq 1$. Then, for $r = 1, \ldots, r_v$ $\mathcal{S}_{FPA}$ waits for (ROUND-WINNER, sid, b_{wr}) and proceeds as follows:
  1. *(honest winner)* If $b_{wr} = 1$ and $b_{ar} = 0$ for each $\mathcal{P}_a \in \mathcal{C}$ (note that $\mathcal{S}_{FPA}$ has previously extracted the bids of the corrupted parties), then $\mathcal{P}_w \notin \mathcal{C}$, *i.e.*, the winner is known to be honest by $\mathcal{S}_{FPA}$ but its identity will be known during Stage 4 only. In this case, $\mathcal{S}_{FPA}$ simulates one honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ picked at random vetoing and simulates all the other honest parties not vetoing again, then goes to Stage 3.
  2. *(corrupted winner)* If $b_{wr} = 1$ and $\mathcal{S}_{FPA}$ receives (LEAK-TO-WINNER, sid, r_w) from $\mathcal{F}_{FPA}$, then $\mathcal{P}_w \in \mathcal{C}$ and its identity is known by $\mathcal{S}_{FPA}$. In this case, $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing and goes to Stage 3.
  3. *(know nothing)* If neither condition 1 nor 2 occurs, then nothing is known yet regarding $\mathcal{P}_w$ (neither its identity nor if $\mathcal{P}_w$ is corrupted or not). If $b_{wr} = 1$ then $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing and goes to Stage 3. If $b_{wr} = 0$ then $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing and continues to simulate Stage 2.
  4. If an honest party $\mathcal{P}_h$ executing Π_{FPA} would initiate the Recovery Stage, $\mathcal{S}_{FPA}$ simulates $\mathcal{P}_h \in \mathcal{H}$ initiate it with the simulated $\mathcal{F}_{\mathsf{SC}}$. Then, at the end of each round r, $\mathcal{S}_{FPA}$ receives (ABORT?, sid) from $\mathcal{F}_{FPA}$. If any $\mathcal{P}_a \in \mathcal{C}$ has been identified as a cheater in the Recovery Stage, $\mathcal{S}_{FPA}$ sends (ABORT, $sid, \mathcal{P}_a$) to $\mathcal{F}_{FPA}$, otherwise $\mathcal{S}_{FPA}$ sends (PROCEED, sid) to $\mathcal{F}_{FPA}$.
- **Stage 3 - After First Veto.** For $r = r_v, \ldots, l$ $\mathcal{S}_{FPA}$ waits for (ROUND-WINNER, sid, b_{wr}) and proceeds as follows:
  1. *(honest winner)* If condition 1 of Stage 2 occurred or occurs during Stage 3, $\mathcal{S}_{FPA}$ eventually picks at random one simulated honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ (in case it has not been already picked during Stage 2) and simulates her vetoing if $b_{wr} = 1$ and not vetoing if $b_{wr} = 0$ until $r = l$ and simulates all the other honest parties not vetoing again
  2. *(corrupted winner)* If condition 2 of Stage 2 occurred or occurs during Stage 3, $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing until $r = l$.
  3. *(know nothing)* If neither condition 1 nor 2 occurs, then nothing is known yet regarding $\mathcal{P}_w$ (neither its identity nor if $\mathcal{P}_w$ is corrupted or not).
     If $b_{wr} = 1$ then $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing. If $b_{wr} = 0$ then $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing.
  4. If an honest party $\mathcal{P}_h$ executing protocol Π_{FPA} would initiate the Recovery Stage, $\mathcal{S}_{FPA}$ simulates $\mathcal{P}_h \in \mathcal{H}$ initiating it with the simulated $\mathcal{F}_{\mathsf{SC}}$. Then, at the end of each round r, $\mathcal{S}_{FPA}$ receives (ABORT?, sid) from $\mathcal{F}_{FPA}$. If any $\mathcal{P}_a \in \mathcal{C}$ has been identified as a cheater in the Recovery Stage, $\mathcal{S}_{FPA}$ sends (ABORT, $sid, \mathcal{P}_a$) to $\mathcal{F}_{FPA}$, otherwise $\mathcal{S}_{FPA}$ sends (PROCEED, sid) to $\mathcal{F}_{FPA}$.

Figure 3.11: Simulator $\mathcal{S}_{FPA}$ (Stages 2 and 3).

Simulator $\mathcal{S}_{FPA}$ (Stages 4 and Recovery)

- **Stage 4 - Output.** Once $\mathcal{S}_{FPA}$ receives (ANNOUNCE-WINNER-AND-REFUND, $sid, \mathcal{P}_w, b_w, \mathsf{coins}(b_i + work)$) from $\mathcal{F}_{FPA}$:
  - If $\mathcal{P}_w \in \mathcal{C}$ and $\mathcal{P}_w$ deviates from the protocol during Stage 4, then $\mathcal{S}_{FPA}$ simulates the recovery procedure with the simulated $\mathcal{F}_{\mathsf{SC}}$. After $\mathcal{P}_w$ halts, $\mathcal{S}_{FPA}$ outputs whatever $\mathcal{P}_w$ outputs.
  - If $\mathcal{P}_w \in \mathcal{C}$ but $\mathcal{P}_w$ does not deviate from the protocol, then the execution is is concluded as in the real world.
  - If $\mathcal{P}_w \notin \mathcal{C}$ then the winning party $\mathcal{P}_w$ in the real world is one of the honest parties. In this case, $\mathcal{S}_{FPA}$ simulates $\mathcal{P}_w \in \mathcal{H}$ winning and opening her commitment c_w not to her dummy input bid b'_w but to the actual bid b_w received from $\mathcal{S}_{FPA}$. Note that $\mathcal{P}_w \in \mathcal{H}$ is, in general, different from the simulated winning honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ that was picked at random during the simulation of Stage 2 or 3. $\mathcal{S}_{FPA}$ uses trapdoor t from Stage 1 - Setup to find a randomness r' such that $c_w = \mathsf{com}(b_w, r')$ by solving $b'_w + t \cdot r_w = b_w + t \cdot r'$ for r'.
- **Recovery Stage.** $\mathcal{S}_{FPA}$ perfectly emulates $\mathcal{F}_{\mathsf{SC}}$. In particular, in case $\mathcal{P}_w$ is corrupted and did not send (OUTPUT, $sid, \mathcal{P}_w, b_w, r_{b_w}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]}$) to $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{S}_{FPA}$ identifies $\mathcal{P}_w$ anyway by observing the extracted inputs bids b_a from each $\mathcal{P}_a \in \mathcal{C}$. Then, $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ sending (RECOVERY-PAYMENT, sid, NW_h) to $\mathcal{F}_{\mathsf{SC}}$ where $NW_h \leftarrow NW\{(V_1 = 1 \wedge v_{h1} = Y_{h1}^{x_{h1}}) \vee \ldots \vee (V_l = 1 \wedge v_{h1} = Y_{k1}^{x_{h1}})\}$ is generated using the simulator for NW. Moreover, $\mathcal{S}_{FPA}$ simulates each committee member $\mathcal{C}_j \in \mathcal{C}_{\mathcal{H}}$ by following $\Pi_{\mathcal{C}}$. Indeed, upon receiving (OPEN, $sid, \mathcal{P}_a$), where $\mathcal{P}_a \in \mathcal{C}$, from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{S}_{FPA}$ simulates $\mathcal{C}_j$ using the share decryption procedure from π_{PVSS} on $\hat{\sigma}_{aj}$, obtaining $\tilde{\sigma}_{aj}, DLEQ_{aj}$. and sending (SHARE-DECRYPTION, $sid, (\hat{\sigma}_{a1}, \ldots, \hat{\sigma}_{am}), LDEI_a, CC_a, \tilde{\sigma}_{aj}, DLEQ_{aj}$) to $\mathcal{F}_{\mathsf{SC}}$.

Figure 3.12: Simulator $\mathcal{S}_{FPA}$ (Stages 4 and Recovery).

simulates one honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ picked at random vetoing or not vetoing according if $b_{wr} = 1$ or $b_{wr} = 0$ respectively and simulates all the other honest parties not vetoing again. Similarly, if nothing is known about the winner, $\mathcal{S}_{FPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing or not vetoing according if $b_{wr} = 1$ or $b_{wr} = 0$ respectively. We will argue why the view of the adversary $\mathcal{A}$ in the simulation by $\mathcal{S}_{FPA}$ is indistinguishable from the real world execution.

By Lemma 4, due to the zero knowledge property of NIZKs, BV_{hr} and AV_{hr} are indistinguishable from the NIZKs of the corresponding parties in the real world. Moreover, by Lemma 5 and Lemma 6, it is proven that the inputs v_{hr} of the veto protocol and the output V_r of each round r are indistinguishable from the inputs of the corresponding parties and the output in the real world. On the other hand, $\mathcal{S}_{FPA}$ can compare the extracted bids b_a of each $\mathcal{P}_a \in \mathcal{C}$ with the output of each round V_r to discover if one of the honest parties in the real world in the winner of

the auction. In that case, as described in the simulator, $\mathcal{S}_{FPA}$ simulates one honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ picked at random behaving as the winner. However, by Lemma 5 and Lemma 6, the adversary $\mathcal{A}$ cannot distinguish which honest party is the winner.

- **Stage 4 - Output:** in case one of the honest parties in the real world is the winner of the auction, in the output stage she will reveal her identity and open the commitment to her bid $\mathsf{com}(b_w, r_{b_w})$ towards the smart contract by sending $(\text{OUTPUT}, sid, \mathcal{P}_w, b_w, r_{b_w}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(b_w)\}_{k\in[n]})$ to $\mathcal{F}_{\mathsf{SC}}$. In the ideal world, as described in the simulator, $\mathcal{S}_{FPA}$ has to simulate $\mathcal{P}_w \in \mathcal{H}$ winning and opening her commitment c_w not to her dummy input bid b'_w but to the bid b_w of $\mathcal{P}_w$ in the real world, *i.e.*, equivocate the commitment. Note that $\mathcal{P}_w \in \mathcal{H}$ is, in general, different from the simulated winning honest party $\mathcal{P}_{h_w} \in \mathcal{H}$ that was picked at random during the simulation of Stage 2 or 3. However, by Lemma 5 and Lemma 6, the adversary $\mathcal{A}$ cannot distinguish if $\mathcal{P}_w$ is different from $\mathcal{P}_{h_w}$.

  Moreover, by Lemma 1, due to the unconditionally hiding property of the commitment, the adversary cannot learn that c_w initially was a commitment to b'_w instead of b_w.

- **Recovery:** $\mathcal{S}_{FPA}$ simulates aborts and corresponding recovery stages if $\mathcal{A}$ deviates from the protocol as in an actual execution of Π_{FPA}, *i.e.*, when an honest party would have triggered the Recovery Stage. Moreover, by Lemma 4, due to the zero-knowledge property of NIZKs, NW_h for each $\mathcal{P}_h \in \mathcal{H}$ are indistinguishable corresponding NIZKs in the real world. Finally, by Proposition 1, it is guaranteed that the shares reconstruction is valid.

Hence, the view of $\mathcal{A}$ in the real world and of $\mathcal{S}_{FPA}$'s internal copy of $\mathcal{A}$ is indistinguishable, which concludes our proof.

□

3.7 Extension to Second-price Auctions

The second-price sealed-bid auction is a type of auction in which the parties first submit their bids to the auctioneer and then the winner is the party with the highest bid, however the price she pays is the second-highest bid. The importance of the second-price auction is that it is *strategy-proof*, *i.e.*, the best strategy for rational parties is to bid their true valuation of the auctioned goods. Despite this, the sealed second-price auctions may not be applied in certain scenarios due to the trust that has to be put in the auctioneer. In particular, a dishonest auctioneer may manipulate the bids and substitute the second-highest bid with a bid that is slightly smaller than the first bid, so as to increase her revenue or disclose the losing bids of the other parties to have a financial gain. In fact, in a recent study, [7] it is shown that the only auction in which the auctioneer has no incentive to deviate from the rules is the first-price auction.

Functionality $\mathcal{F}_{SPA}$

$\mathcal{F}_{SPA}$ operates with an auctioneer $\mathcal{P}_{AUC}$, a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ who have bids $b_1, \ldots, b_n$ as input, as well as an adversary $\mathcal{S}_{SPA}$. $\mathcal{F}_{SPA}$ is parameterized by a bid bit-length l and keeps an initially empty list bids.

- **Setup (Bid Registration):** Upon receiving $(\text{BID}, sid, \mathsf{coins}(b_i + work))$ from $\mathcal{P}_i$ where $b_i \in \{0,1\}^l$ and $work$ is the amount required to compensate the cost of running the protocol for all the other parties, $\mathcal{F}_{SPA}$ appends b_i to bids.
- **Winner Selection:** After receiving $(\text{BID}, sid, \mathsf{coins}(b_i + work))$ from all parties in $\mathcal{P}$, for $r = 1, \ldots, r_w$, where r_w is initialized to l, $\mathcal{F}_{SPA}$ proceeds as follows:
  1. Select b_{wr}, *i.e.*, the r-th bit of the highest bid b_w in the list bids.
  2. Send $(\text{ROUND-WINNER}, sid, b_{wr})$ to all parties.
  3. Check if $b_{wr} = 1$ and $b_{ir} = 0$ for $i = 1, \ldots, n \neq w$. If so, set $r_w = r$, that is the first position where b_w has a bit 1 and b_{w_2} has a bit 0, and send $(\text{LEAK-TO-WINNER}, sid, r_w)$ to $\mathcal{P}_w$.
  4. If $\mathcal{P}_w$ is honest, announce her identity by sending $(\text{ANNOUNCE-WINNER}, sid, \mathcal{P}_w)$ to all $\mathcal{P}_i \in \mathcal{P}$ and $\mathcal{S}_{SPA}$. If $\mathcal{P}_w$ is corrupted, send $(\text{ANNOUNCE?}, sid, \mathcal{P}_w)$ to $\mathcal{S}_{SPA}$. If $\mathcal{S}_{SPA}$ answers with $(\text{ANNOUNCE}, sid)$ then send $(\text{ANNOUNCE-WINNER}, sid, \mathcal{P}_w)$ to all $\mathcal{P}_i \in \mathcal{P}$. If $\mathcal{S}_{SPA}$ answers with $(\text{NOT-ANNOUNCE}, sid)$, do nothing.
  5. Send $(\text{ABORT?}, sid)$ to $\mathcal{S}_{SPA}$. If $\mathcal{S}_{SPA}$ answers with $(\text{ABORT}, sid, \mathcal{P}_i)$ where $\mathcal{P}_i$ is corrupted, remove b_i from bids, remove $\mathcal{P}_i$ from $\mathcal{P}$, send $(\text{ABORT}, sid, \mathcal{P}_i, \mathsf{coins}(\frac{b_i + work}{|\mathcal{P}|} + work))$ where $|\mathcal{P}|$ is the number of remaining parties to all other parties in $\mathcal{P}$ to $\mathcal{S}_{SPA}$, set again $r = 1$ and go to Step 1. If $\mathcal{S}_{SPA}$ answers with $(\text{PROCEED}, sid)$, if $\mathcal{P}_w$ has been determined, *i.e.*, $b_{wr} = 1$ and $b_{ir} = 0$ for $i = 1, \ldots, n \neq w$, go to Second-Price Selection, otherwise increment r by 1 and go to Step 1.
- **Second-Price Selection:** remove $\mathcal{P}_w$ from $\mathcal{P}$ and for $r = r_w, \ldots, l$ $\mathcal{F}_{SPA}$ proceeds as follows:
  1. Select $b_{w_2 r}$, *i.e.*, the r-th bit of the second highest bid b_{w_2} in the list bids.
  2. Send $(\text{ROUND-WINNER}, sid, b_{w_2 r})$ to all parties.
  3. Check if $b_{w_2 r} = 1$ and $b_{ir} = 0$ for $i = 1, \ldots, n \neq w_2$. If so, set $r_{w_2} = r$, that is the first position where b_{w_2} has a bit 1 and b_{w_3} has a bit 0, and send $(\text{LEAK-TO-SECOND}, sid, r_{w_2})$ to $\mathcal{P}_{w_2}$.
  4. Send $(\text{ABORT?}, sid)$ to $\mathcal{S}_{SPA}$. If $\mathcal{S}_{SPA}$ answers with $(\text{ABORT}, sid, \mathcal{P}_i)$ where $\mathcal{P}_i$ is corrupted, remove b_i from bids, remove $\mathcal{P}_i$ from $\mathcal{P}$, send $(\text{ABORT}, sid, \mathcal{P}_i, \mathsf{coins}(\frac{b_i + work}{|\mathcal{P}|}))$ where $|\mathcal{P}|$ is the number of remaining parties to all other parties in $\mathcal{P}$, set again $r = r_w$ and go to Step 1 to start recomputing the second-price. If $\mathcal{S}_{SPA}$ answers with $(\text{PROCEED}, sid)$, if $\mathcal{P}_{w_2}$ has been determined, *i.e.*, $b_{w_2 r} = 1$ and $b_{ir} = 0$ for $i = 1, \ldots, n \neq w_2$, go to Payout, otherwise increment r by 1 and go to Step 1.
- **Payout:** Send $(\text{REFUND}, sid, \mathsf{coins}(b_i + work))$ to all parties $\mathcal{P}_i \neq \mathcal{P}_w$, send $(\text{REFUND}, sid, \mathsf{coins}(work))$ to $\mathcal{P}_w$, send $\mathsf{coins}(b_{w_2})$ to $\mathcal{P}_{AUC}$, and halt.

Figure 3.13: Functionality $\mathcal{F}_{SPA}$.

Hence, when considering the second-price we must overcome these problems. We propose an efficient solution to adapt our protocol in the case of second-price auctions. Note that a trivial solution is to run the protocol for the first-price twice, but the second time from Stage 2 using the same setup from Stage 1 and without the winning party $\mathcal{P}_w$. However, this discloses both the highest bid b_w and the second-highest bid b_{w_2} (*i.e.*, the price paid by the winning party), and suffers of unnecessary computational and communication complexity.

Modelling Second-Price Fair Auctions:

First, we describe an ideal functionality $\mathcal{F}_{SPA}$ for the second-price auctions we realize in Figure 3.13.

The Protocol:

Protocol Π_{SPA} for Second-Price Auctions is described in Figures 3.14 and 3.15. In this protocol, each party $\mathcal{P}_i$ checks if she is the only one veto-ing in every round r where $b_{ir} = 1$ and $V_r \neq 1$ (*i.e.*, in which there was a veto), which means that $\mathcal{P}_i$ is the winning party. Each party $\mathcal{P}_i$ can do that by checking whether she obtains an alternative value $V'_r = 1$ (no veto) by using $v'_{ir} = Y_{ir}^{x_{ir}}$ (no veto) as her message and keeping the other parties' messages unchanged. If this condition is satisfied, then $\mathcal{P}_i$ proves it to all the other parties by revealing x_{ir}. The first party who proves this condition to be true is the winning party $\mathcal{P}_w$. In order to compute the second-highest bid b_{w_2}, the other parties conclude the protocol excluding $\mathcal{P}_w$, which reduces drastically both computational and communication complexity with respect to the trivial solution of re-executing the protocol for the first-price from scratch without $\mathcal{P}_w$. In fact, the complexity of the protocol for the second-price is almost the same as the one for the first-price case, *i.e.*, when $\mathcal{P}_w$ sends x_{ir} to all the other parties, so as to prove she is the winning party, the communication complexity increases by $|\mathbb{Z}_q|$ only. We present a detailed efficiency estimate in Section 3.8. Finally, by using this approach the winning bid is just partially disclosed, *i.e.*, the knowledge of the round r in which $\mathcal{P}_w$ declared herself as the winner of the auctions provides a lower bound only over her actual bid b_w.

Security Analysis:

The security of Protocol Π_{SPA} is stated in Theorem 2 and proven in Section 3.7.1. A game-theoretical analysis is presented in Section 3.9.

Theorem 2. *Under the DDH Assumption, Protocol Π_{SPA} securely computes $\mathcal{F}_{SPA}$ in the $\mathcal{F}_{\mathsf{SC}}$-hybrid, random oracle model against a malicious static adversary $\mathcal{A}$ corrupting all but one parties $\mathcal{P}_i \in \mathcal{P}$ and $m/2 - 2$ parties $\mathcal{C}_i \in \mathcal{C}$.*

3.7.1 Proof of Theorem 2

In this section, we prove Theorem 2, which we reproduce below for the sake of clarity.

Protocol Π_{SPA} (Stages 1, 2, 3a and 3b)

Protocol Π_{SPA} is executed by parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, where each party $\mathcal{P}_i$ has a l-bit bid $b_i = b_{i1}|\ldots|b_{il}$, and a deposit committee $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ interacting among themselves and with a smart contract $\mathcal{F}_{\mathsf{SC}}$.

Off-chain messages exchange and Stage 1 - Setup: Same as in Protocol Π_{FPA}.

Stage 2 - Before First Veto and Stage 3a - After First Veto: In this stage, all parties $\mathcal{P}_i$ follow all the steps of Stage 2 and Stage 3 of Π_{FPA} respectively and execute the following extra steps:

1. If $\mathcal{P}_i$ sent $v_{ir} = g^{\bar{r}_{ir}}$ (*i.e.*, she has $b_{ir} = 1$ and issued a veto), $\mathcal{P}_i$ computes $V'_r = \big(\prod_{k=1,k\neq i}^{n} v_{kr}\big) Y_{ir}^{x_{ir}}$ and checks whether $V_r \neq 1$ and $V'_r = 1$. If this is true, it means that $\mathcal{P}_i$ is the only one who has vetoed (*i.e.*, she is the only party with $b_{kr} = 1$ for $k = 1, \ldots, n$, implying she has the highest bid). In this case, $\mathcal{P}_i$ sends (WINNER, $sid, \mathcal{P}_w, x_{ir}$) to all other parties.
2. Upon receiving (WINNER, $sid, \mathcal{P}_w, x_{ir}$) from $\mathcal{P}_w$, $\mathcal{P}_i$ it checks whether $\mathcal{P}_w$ indeed has the highest bid by checking that $X_{wr} = g^{x_{wr}}$, computing $V'_r = \big(\prod_{k=1,k\neq w}^{n} v_{wr}\big) Y_{wr}^{x_{wr}}$ and checking whether $V_r \neq 1$ and $V'_r = 1$. If any of these checks fail, proceed to the Recovery Stage.
3. If a valid message (WINNER, $sid, \mathcal{P}_w, x_{ir}$) was $\mathcal{P}_w$, all parties $\mathcal{P}_i \in \mathcal{P} \setminus \mathcal{P}_w$ recompute $Y'_{kr} = \prod_{m=1}^{k-1} X_{mr} / \prod_{m=k+1}^{n} X_{mr}$ for $k \in \{1, \ldots, n\} \setminus w$, $k = j+1, \ldots, l$ (*i.e.*, excluding $\mathcal{P}_w$ from the remaining rounds from $j+1$ to l), then continue to Stage 3b by executing the protocol using the new Y'_{ir} with a set of parties $\mathcal{P} \setminus \mathcal{P}_w$ excluding $\mathcal{P}_w$.
4. If no party has sent a message (WINNER, $sid, \mathcal{P}_w, x_{ir}$) by round $r = l$, then the winner is dishonest (assuming no tie) and $\mathcal{P}_i$ proceeds to the Recovery Stage.

Stage 3b - After First Unique Veto: In this stage, all parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \setminus \mathcal{P}_w$ considers $V_r = V'_r = \big(\prod_{k=1,k\neq w}^{n} v_{wr}\big) Y_{wr}^{x_{wr}} = 1$ (*i.e.*, V_r with the input v_{wr} of $\mathcal{P}_w$ not representing a veto instead). Then:

- If there does not exist another previous round z with $z = 1, \ldots, r-1$ such that $V_z \neq 1$, *i.e.*, the first veto was also the first unique veto, then the parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \setminus \mathcal{P}_w$ continue the protocol following all the steps is Stage 2 and Stage 3 described in Section 3.6 but using the values Y'_{ir} recomputed after the first unique veto was detected. When Stage 2 and eventually Stage 3 are completed, return and go to the output stage described in this section.
- If there does exist another previous round z with $z = 1, \ldots, r-1$ such that $V_z \neq 1$, *i.e.*, the first unique veto was not the first veto, then set $\hat{r} = z$ (*i.e.*, the index of the last veto is changed from r to z) and the parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \setminus \mathcal{P}_w$ continue the protocol following all the steps from Stage 3 as described in Section 3.6 but using $\hat{r} = z$ when computing their next input $v_{i(r+1)}$ and the values Y'_{ir} recomputed after the first unique veto was detected. When Stage 3 is completed, return and go to the output stage described in this section.

Figure 3.14: Protocol Π_{SPA} (Stages 1, 2, 3a and 3b).

Protocol Π_{SPA} (Stage 4 and Recovery Stage)

Stage 4 - Output: At this point, all parties know the winner party $\mathcal{P}_w$ and the second-price b_{w_2}. The protocol proceeds as follows:

1. $\mathcal{P}_i$ computes the second highest bid as $b_{w_2} = b_{w_2 1}|\cdots|b_{w_2 l}$, such that $b_{w_2 r} = 1$ if $V_r \neq 1$ and $b_{w_2 r} = 0$ if $V_r = 1$, and sends $\mathcal{P}_w, b_{w_2}$ to all other parties (causing all parties $\mathcal{P}_k$ to sign b_{w_2} and send $\mathsf{Sig.Sign}_{\mathsf{sk}_k}(\mathcal{P}_w|b_{w_2})$ to each other).
2. In the Setup Stage, $\mathcal{P}_w$ sent to the smart contract a confidential transaction $\mathsf{tx}_w = (\mathtt{id}, \mathsf{In}, \mathsf{Out}, \mathsf{Sig}, r_{b_w} + r_{change_w}, \pi)$ where $\mathsf{Out} = \{(\mathsf{com}(b_w, r_{b_w}), Addr_s), (work, Addr_s), (\mathsf{com}(change_w, r_{change_w}), Addr_w)\}$. $\mathcal{P}_w$ creates a new confidential transaction $\mathsf{tx}_{pay} = (\mathtt{id}_{pay}, \mathsf{In}_{pay}, \mathsf{Out}_{pay}, \mathsf{Sig}_{pay}, r_{b_w} - r_{change'_w}, \pi_{pay})$ where $\mathsf{In} = \{(\mathtt{id}, \mathsf{com}(b_w, r_{b_w}))\}$, $\mathsf{Out} = \{(b_{w_2}, Addr_{auc}), (\mathsf{com}(change'_w, r_{change'_w}), Addr_w)\}$, Sig_{pay} is left empty, $Addr_{auc}$ is the address of the auctioneer, $Addr_w$ is the address of $\mathcal{P}_w$ and π_{pay} is a NIZK showing that $change'_w$ is between $[0, 2^l - 1]$. $\mathcal{P}_w$ sends (OUTPUT, $sid, \mathcal{P}_w, \mathsf{tx}_{pay}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(\mathcal{P}_w|b_{w_2})\}_{k\in[n]}$) to $\mathcal{F}_{\mathsf{SC}}$, which performs tx_{pay} after checking the validity of the message so that the auctioneer receives the payment b_{w_2} and $\mathcal{P}_w$ gets back $b_w - b_{w_2}$.
3. Finally, all honest parties receive a refund of their deposit from the smart contract, apart from the winning party, who only receives a refund of the fee $work$ plus the transaction $(b_w - b_{w_2})$ computed in the previous step.

Recovery Stage: Parties $\mathcal{C}_i \in \mathcal{C}$ listen to $\mathcal{F}_{\mathsf{SC}}$ and execute the Share Decryption step of $\Pi_{\mathcal{C}}$ from Figure 3.3 if requested. In case a party $\mathcal{P}_i \in \mathcal{P}$ is suspected of cheating, the Recovery stage is executed depending on the exact suspected cheating as defined in Protocol Π_{FPA}, which allows to eventually identify the cheater. If a cheater $\mathcal{P}_i$ is identified, the Recovery Stage proceeds as follows:

- **Re-execution (unknown $\mathcal{P}_w$)**: in this scenario the winning the party $\mathcal{P}_w$ is still unknown, then the parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \setminus \mathcal{P}_i$ re-execute the protocol from Stage 2 without the cheating party $\mathcal{P}_i$.
- **Re-execution (known $\mathcal{P}_w$ but unknown b_{w_2})**: in this scenario the winning party $\mathcal{P}_w$ is known but the second highest bid b_{w_2} is unknown, then the parties $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \setminus \{\mathcal{P}_i, \mathcal{P}_w\}$ re-execute the protocol from Stage 2 without the cheating party $\mathcal{P}_i$ and the winning party $\mathcal{P}_w$.
- **Complete payment (known $\mathcal{P}_w$ and b_{w_2} but missing payment)**: in this scenario both the winning party $\mathcal{P}_w$ and the second highest bid b_{w_2} are known, but $\mathcal{P}_w$ is has not completed the payment to the auctioneer. Then, $\mathcal{P}_w$'s deposit $b_w + work$ is distributed by the smart contract as follows: b_{w_2} is sent to the auctioneer and the remaining amount, that is equal to $(b_w + work - b_{w_2})$, is distributed to the other parties.
- **Dishonest winner identification:** If no party has sent a message (WINNER, $sid, \mathcal{P}_w, x_{ir}$) by round $r = l$, each $\mathcal{P}_i \in \mathcal{P}$ computes a NIZK $NW_i \leftarrow NW\{x_{i1}, \ldots, x_{il} \mid (V_1 = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}}) \vee \ldots \vee (V_l = 1 \wedge v_{i1} = Y_{i1}^{x_{i1}})\}$ showing that they are not the winner and sends to $\mathcal{F}_{\mathsf{SC}}$ (RECOVERY-DISHONEST-WINNER, sid, NW_i). All parties $\mathcal{P}_i$ who do not send a valid NW_i are identified as dishonest and have their deposits distributed among the honest parties.

Figure 3.15: Protocol Π_{SPA} (Stage 4 and Recovery Stage).

Simulator $\mathcal{S}_{SPA}$ (Stages 1 and 2)

Let $\mathcal{H}$ be the set of simulated honest parties, $\mathcal{C}$ be the set of parties corrupted by the adversary $\mathcal{A}$, $\mathcal{C}_{\mathcal{H}}$ be the set of simulated members of the committee and $\mathcal{C}_{\mathcal{C}}$ be the set of members of the committee corrupted by the adversary $\mathcal{A}$.

- **Stage 1 - Setup.** Same as in $\mathcal{S}_{FPA}$.

  Simulating an honest party vetoing or not vetoing. Same as $\mathcal{S}_{FPA}$.
- **Stage 2 - Before First Veto.** For every round r before the first veto, $\mathcal{S}_{SPA}$ waits for (ROUND-WINNER, sid, b_{wr}) and proceeds as follows:

  1. *(honest winner)* If $\mathcal{F}_{SPA}$ sends (ANNOUNCE-WINNER, $sid, \mathcal{P}_w$) to $\mathcal{S}_{SPA}$, then the winner is an honest party $\mathcal{P}_w$. Thus, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w$ vetoing and proving to be the winner by sending to all the other parties (WINNER, $sid, \mathcal{P}_w, x_{wr}$) and simulates all the other honest parties not vetoing again, then goes to Stage 3b.
  2. *(corrupted winner)* Upon receiving (LEAK-TO-WINNER, sid, r_w) and subsequently (ANNOUNCE?, $sid, \mathcal{P}_w$) from $\mathcal{F}_{SPA}$, $\mathcal{S}_{SPA}$ learns the identity of the winner $\mathcal{P}_w$ and that is corrupted. If $\mathcal{A}$ sends a message (WINNER, $sid, \mathcal{P}_w, x_{wr}$) then $\mathcal{S}_{SPA}$ replies (ANNOUNCE, sid) to $\mathcal{F}_{SPA}$, otherwise it replies (NOT-ANNOUNCE, sid) to $\mathcal{F}_{SPA}$. Then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing and goes to Stage 3b.
  3. *(know nothing)* If neither condition 2 nor 3 occurs, then nothing is known yet regarding $\mathcal{P}_w$ (neither its identity nor if $\mathcal{P}_w$ is corrupted or not). If $b_{wr} = 1$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing and goes to Stage 3a. If $b_{wr} = 0$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing and continues to simulate Stage 2.
  4. If an honest party $\mathcal{P}_h$ executing Π_{SPA} would initiate the Recovery Stage, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_h$ initiating it with the simulated $\mathcal{F}_{\mathsf{SC}}$. Then, at the end of each round r, $\mathcal{S}_{SPA}$ receives (ABORT?, sid) from $\mathcal{F}_{SPA}$. If any $\mathcal{P}_a \in \mathcal{C}$ has been identified as a cheater in the Recovery Stage, $\mathcal{S}_{SPA}$ sends (ABORT, $sid, \mathcal{P}_a$) to $\mathcal{F}_{SPA}$, otherwise $\mathcal{S}_{SPA}$ sends (PROCEED, sid) to $\mathcal{F}_{SPA}$.

Figure 3.16: Simulator $\mathcal{S}_{SPA}$ (Stages 1 and 2).

Theorem 2. *Under the DDH Assumption, Protocol Π_{SPA} securely computes $\mathcal{F}_{SPA}$ in the $\mathcal{F}_{\mathsf{SC}}$-hybrid, random oracle model against a malicious static adversary $\mathcal{A}$ corrupting all but one parties $\mathcal{P}_i \in \mathcal{P}$ and $m/2 - 2$ parties $\mathcal{C}_i \in \mathcal{C}$.*

Proof. In order to prove this theorem, we construct a simulator $\mathcal{S}_{SPA}$ (Figure 3.16, Figure 3.17, Figure 3.18) that performs an ideal execution with $\mathcal{F}_{SPA}$ and interacts with an internal copy of the adversary $\mathcal{A}$, simulates honest parties, $\mathcal{F}_{\mathsf{SC}}$ and the random oracle in an execution of Protocol Π_{SPA} with $\mathcal{A}$ in such a way that this execution is indistinguishable from an execution between $\mathcal{A}$ and an honest party in the real world.

Throughout this execution, $\mathcal{S}_{SPA}$ perfectly emulates $\mathcal{F}_{\mathsf{SC}}$ and the random oracle unless stated otherwise. In order to show that an ideal execution with $\mathcal{S}_{SPA}$ and $\mathcal{F}_{SPA}$ is indistinguishable from a real execution of Π_{SPA} with $\mathcal{A}$ and honest parties, we argue that the view of $\mathcal{A}$ in the real world and of $\mathcal{S}_{SPA}$'s internal copy of $\mathcal{A}$ is indistinguishable. In particular:

Simulator $\mathcal{S}_{SPA}$ (Stage 3a and 3b)

- **Stage 3a - After First Veto** For every round r after the first veto, $\mathcal{S}_{SPA}$ waits for (ROUND-WINNER, sid, b_{wr}) and proceeds as follows:
  1. *(honest winner)* If $\mathcal{F}_{SPA}$ sends (ANNOUNCE-WINNER, $sid, \mathcal{P}_w$) to $\mathcal{S}_{SPA}$, then the winner is an honest party $\mathcal{P}_w$. Thus, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w \in \mathcal{H}$ vetoing and proving to be the winner by sending to all the other parties x_{wr}, simulates all the other honest parties $\mathcal{H} \setminus \mathcal{P}_w$ not vetoing, then goes to Stage 3b.
  2. *(corrupted winner)* Upon receiving (LEAK-TO-WINNER, sid, r_w) and subsequently (ANNOUNCE?, $sid, \mathcal{P}_w$) from $\mathcal{F}_{SPA}$, $\mathcal{S}_{SPA}$ learns the identity of the winner $\mathcal{P}_w$ and that is corrupted. If $\mathcal{A}$ sends a message (WINNER, $sid, \mathcal{P}_w, x_{wr}$) then $\mathcal{S}_{SPA}$ replies (ANNOUNCE, sid) to $\mathcal{F}_{SPA}$, otherwise it replies (NOT-ANNOUNCE, sid) to $\mathcal{F}_{SPA}$. Then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing and goes to Stage 3b.
  3. *(know nothing)* If neither condition 2 nor 3 occurs, then nothing is known yet regarding $\mathcal{P}_w$ (neither its identity nor if $\mathcal{P}_w$ is corrupted or not). If $b_{wr} = 1$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing. If $b_{wr} = 0$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing.
  4. If an honest party $\mathcal{P}_h$ executing Π_{SPA} would initiate the Recovery Stage, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_h$ initiating it with the simulated $\mathcal{F}_{\mathsf{SC}}$. Then, at the end of each round r, $\mathcal{S}_{SPA}$ receives (ABORT?, sid) from $\mathcal{F}_{SPA}$. If any $\mathcal{P}_a \in \mathcal{C}$ has been identified as a cheater in the Recovery Stage, $\mathcal{S}_{SPA}$ sends (ABORT, $sid, \mathcal{P}_a$) to $\mathcal{F}_{SPA}$, otherwise $\mathcal{S}_{SPA}$ sends (PROCEED, sid) to $\mathcal{F}_{SPA}$.
- **Stage 3b - After First Unique Veto.** If $\mathcal{P}_w$ is honest, $\mathcal{S}_{SPA}$ excludes it from Stage 3b. For all other rounds r, $\mathcal{S}_{SPA}$ waits for (ROUND-WINNER, sid, b_{w_2r}) and:
  1. *(honest second)* If $b_{w_2r} = 1$ and $b_{ar} = 0$ for each $\mathcal{P}_a \in \mathcal{C}$ (note that $\mathcal{S}_{SPA}$ has previously extracted the bids of the corrupted parties), then $\mathcal{S}_{SPA}$ learns that the second-price bid was done by an honest party $\mathcal{P}_{w_2}$. In this case, $\mathcal{S}_{SPA}$ simulates one honest party $\mathcal{P}_{h_{w_2}}$ picked at random (excluding the 1st price winner $\mathcal{P}_w$ if it is honest) vetoing if $b_{w_2r} = 1$ and not vetoing if $b_{w_2r} = 0$ until $r = l$ and simulates all the other honest parties not vetoing again.
  2. *(corrupted second)* If $b_{w_2r} = 1$ and $\mathcal{S}_{SPA}$ receives (LEAK-TO-SECOND, sid, r_{w_2}) from $\mathcal{F}_{SPA}$, then $\mathcal{S}_{SPA}$ learns that the second-price bid belongs to a corrupted party $\mathcal{P}_{w_2} \in \mathcal{C}$. In this case, $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h$ not vetoing until $r = l$.
  3. *(know nothing)* If neither condition 2 nor 3 occurs, then nothing is known yet regarding $\mathcal{P}_{w_2}$ (neither its identity nor if $\mathcal{P}_{w_2}$ is corrupted or not). If $b_{w_2r} = 1$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ vetoing. If $b_{w_2r} = 0$ then $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ not vetoing. Then continues to simulate Stage 3b until $r = l$.
  4. If an honest party $\mathcal{P}_h$ executing Π_{SPA} would initiate the Recovery Stage, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_h \in \mathcal{H}$ initiating it with the simulated $\mathcal{F}_{\mathsf{SC}}$. Then, at the end of each round r, $\mathcal{S}_{SPA}$ receives (ABORT?, sid) from $\mathcal{F}_{SPA}$. If any $\mathcal{P}_a \in \mathcal{C}$ has been identified as a cheater in the Recovery Stage, $\mathcal{S}_{SPA}$ sends (ABORT, $sid, \mathcal{P}_a$) to $\mathcal{F}_{SPA}$, otherwise $\mathcal{S}_{SPA}$ sends (PROCEED, sid) to $\mathcal{F}_{SPA}$.

Figure 3.17: Simulator $\mathcal{S}_{SPA}$ (Stage 3a and 3b).

Simulator $\mathcal{S}_{SPA}$ (Stage 4 and Recovery)

- **Stage 4 - Output.** Once $\mathcal{S}_{SPA}$ receives $(\text{REFUND}, sid, b_{w_2}, \mathsf{coins}(b_i + work))$ from $\mathcal{F}_{SPA}$:
  - If $\mathcal{P}_w$ is corrupted and deviates from the protocol during Stage 4, then $\mathcal{S}_{SPA}$ simulates the recovery procedure with the simulated $\mathcal{F}_{\mathsf{SC}}$.
  - If $\mathcal{P}_w$ is corrupted but does not deviate from the protocol, $\mathcal{S}_{SPA}$ follows the instructions of an honest party in Π_{SPA} to simulate honest parties.
  - If $\mathcal{P}_w$ is honest, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w$ winning by creating a confidential transaction $\mathsf{tx}_{pay} = (\mathtt{id}_{pay}, \mathsf{In}_{pay}, \mathsf{Out}_{pay}, \mathsf{Sig}_{pay}, r_{b_w} - r_{change'_w}, \pi_{pay})$ where $\mathsf{In} = \{(\mathtt{id}, \mathsf{com}(b_w, r_{b_w}))\}$, $\mathsf{Out} = \{(b_{w_2}, Addr_{auc}), (\mathsf{com}(change'_w, r_{change'_w}), Addr_w)\}$, Sig_{pay} is left empty, $Addr_{auc}$ is the address of the auctioneer, $Addr_w$ is the address of $\mathcal{P}_w$ and π_{pay} is a NIZK showing that $change'_w$ is within $[0, 2^l - 1]$ (using the simulator for this NIZK). Finally, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w$ sending $(\text{OUTPUT}, sid, \mathcal{P}_w, \mathsf{tx}_{pay}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(\mathcal{P}_w | b_{w_2})\}_{k\in[n]})$ to $\mathcal{F}_{\mathsf{SC}}$.
  - After $\mathcal{P}_w$ halts, $\mathcal{S}_{SPA}$ outputs whatever $\mathcal{P}_w$ outputs.
- **Recovery Stage.** $\mathcal{S}_{SPA}$ perfectly emulates $\mathcal{F}_{\mathsf{SC}}$. In particular, in case $\mathcal{P}_w$ is corrupted and did not send $(\text{WINNER}, sid, \mathcal{P}_w, x_{wr})$ to all the other parties, $\mathcal{S}_{SPA}$ knows her identity due to the message $(\text{ANNOUNCE?}, sid, \mathcal{P}_w)$ from $\mathcal{F}_{SPA}$. Then, $\mathcal{S}_{SPA}$ simulates all honest parties $\mathcal{P}_h \in \mathcal{H}$ sending $(\text{RECOVERY-DISHONEST-WINNER}, sid, NW_h)$ to $\mathcal{F}_{\mathsf{SC}}$ where $NW_h \leftarrow NW\{(V_1 = 1 \wedge v_{h1} = Y_{h1}^{x_{h1}}) \vee \ldots \vee (V_l = 1 \wedge v_{h1} = Y_{k1}^{x_{h1}})\}$ is generated using the simulator of NW. Moreover, $\mathcal{S}_{SPA}$ simulates each committee member $\mathcal{C}_j \in \mathcal{C}_{\mathcal{H}}$ by following $\Pi_{\mathcal{C}}$. Indeed, upon receiving $(\text{OPEN}, sid, \mathcal{P}_a)$, where $\mathcal{P}_a \in \mathcal{C}$, from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{S}_{SPA}$ simulates $\mathcal{C}_j$ using the share decryption procedure from π_{PVSS} on $\hat{\sigma}_{aj}$, obtaining $\tilde{\sigma}_{aj}, DLEQ_{aj}$. and sending $(\text{SHARE-DECRYPTION}, sid, (\hat{\sigma}_{a1}, \ldots, \hat{\sigma}_{am}), LDEI_a, CC_a, \tilde{\sigma}_{aj}, DLEQ_{aj}))$ to $\mathcal{F}_{\mathsf{SC}}$.

Figure 3.18: Simulator $\mathcal{S}_{SPA}$ (Stage 4 and Recovery).

- **Stage 1 - Setup:** Same as Theorem 1.
- **Stage 2 - Before First Veto and Stage 3a - After First Veto:** In terms of differences with respect of $\mathcal{S}_{FPA}$, in case an honest party is known to be the winner in a certain round r, $\mathcal{S}_{SPA}$ immediately learns the identity of the winning party by receiving the message $(\text{ANNOUNCE-WINNER}, sid, \mathcal{P}_w)$ from $\mathcal{F}_{SPA}$. Then $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w \in \mathcal{H}$ vetoing and proving to be the winner by sending to all the other parties $(\text{WINNER}, sid, \mathcal{P}_w, x_{wr})$ and simulates all the other honest parties not vetoing again, then goes to Stage 3b. Notice that, since $x_{wr} \xleftarrow{\$} \mathbb{Z}_q$ is sampled as in Π_{SPA}, this step is indistinguishable from that as in real world execution of Π_{SPA}. On the other hand, upon receiving $(\text{LEAK-TO-WINNER}, sid, r_w)$ and subsequently $(\text{ANNOUNCE?}, sid, \mathcal{P}_w)$ from $\mathcal{F}_{SPA}$, $\mathcal{S}_{SPA}$ learns the identity of the winner $\mathcal{P}_w$ and that is corrupted. Then, in case $\mathcal{P}_w$ does not send $(\text{WINNER}, sid, \mathcal{P}_w, x_{wr})$ to all the other parties, $\mathcal{S}_{SPA}$ sends $(\text{NOT-ANNOUNCE}, sid,)$ to $\mathcal{F}_{SPA}$ and $\mathcal{P}_w$ will be identified as a cheater when the recovery procedure is simulated. Analogous consideration to the Theorem 1 case (Stages

2 and 3) and this approach make the view of the adversary $\mathcal{A}$ in the simulation by $\mathcal{S}_{SPA}$ indistinguishable from the real world execution.

- **3b - After First Unique Veto:** At this point, in case $\mathcal{P}_w \in \mathcal{C}$, she will not participate in the next steps of Stage 3b. Similarly, in case $\mathcal{P}_w \notin \mathcal{C}$, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w \in \mathcal{H}$ not participating in the next steps of Stage 3b. Then, the second-price b_{w_2} has to be determined. Analogous consideration to the Theorem 1 case (Stage 3) and this approach make the view of the adversary $\mathcal{A}$ in the simulation by $\mathcal{S}_{SPA}$ indistinguishable from the real world execution.

- **Stage 4 - Output:** In case one of the honest parties in the real world is the winner, $\mathcal{S}_{SPA}$ simulates $\mathcal{P}_w \in \mathcal{H}$ winning by creating a confidential transaction tx_{pay} and sending $(\text{OUTPUT}, sid, \mathcal{P}_w, \mathsf{tx}_{pay}, \{\mathsf{Sig.Sign}_{\mathsf{sk}_k}(\mathcal{P}_w|b_{w_2})\}_{k\in[n]})$ to $\mathcal{F}_{\mathsf{SC}}$. However, by Lemma 1, due to the hiding property of the commitments, the distributions of $\mathsf{com}(b_w, r_{b_w})$ and $\mathsf{com}(change'_w, r_{change'_w})$ are indistinguishable from those in a real world execution of Π_{SPA}. Moreover, by Lemma 3, due to the zero knowledge property of range NIZKs, π_{pay} is indistinguishable from the range proof of $\mathcal{P}_w$ in the real world.

  On the other hand, in the case $\mathcal{P}_w \in \mathcal{C}$, by Lemma 1, due to the binding property of the commitments, the committed values b_w and $change'_w$ cannot be changed later by $\mathcal{A}$. Moreover, by Lemma 3, due to the soundness property of the NIZKs, it is computationally hard for the adversary $\mathcal{A}$ controlling each $\mathcal{P}_a \in \mathcal{C}$ to compute the range proof π_{pay} while the bid is not in the expected range.

- **Recovery:** $\mathcal{S}_{SPA}$ simulates aborts and corresponding recovery stages if $\mathcal{A}$ deviates from the protocol following the instructions of an honest party executing Π_{SPA}. Moreover, by Lemma 4, due to the zero knowledge property of NIZKs, NW_h for each $\mathcal{P}_h \in \mathcal{H}$ are indistinguishable from the corresponding NIZKs in the real world. Finally, by Proposition 1, it is guaranteed that the shares reconstruction is valid.

Hence, the view of $\mathcal{A}$ in the real world and of $\mathcal{S}_{SPA}$'s internal copy of $\mathcal{A}$ is indistinguishable, which concludes our proof.

□

	Stage 1	Stage 2	Stage 3	Total
FAST	$nl + l + 8\log l + 2$	$\tau(8 + 10n)$	$(l-\tau)(19+22n)$	$23nl+20l+8\log l - 11\tau - 12n\tau + 2$
SEAL [13]	$11l + 12nl$	$\tau(17 + 20n)$	$(l-\tau)(33+36n)$	$48nl + 44l - 16\tau - 16n\tau$

Table 3.1: First-price auction computational complexity comparison in terms of exponentiations performed by a party $\mathcal{P}_i \in \mathcal{P}$: n is the number of parties, l is the total number of rounds in Stages 2 and 3 (*i.e.*, bit-length of bids), τ is the number of rounds in Stage 2.

	Stage 1	Stage 2	Stage 3	Total
FAST	$n((2l + 10)\lvert\mathbb{G}\rvert + 3\lambda + 4\log l)$	$n\tau(\lvert\mathbb{G}\rvert + 6\lvert\mathbb{Z}_q\rvert)$	$n(l-\tau)(\lvert\mathbb{G}\rvert + 11\lvert\mathbb{Z}_q\rvert)$	$n\big(\lvert\mathbb{G}\rvert(3l + 10) + \lvert\mathbb{Z}_q\rvert(11l - 5\tau) + 3\lambda + 4\log l\big)$
SEAL [13]	$17nl\lvert\mathbb{G}\rvert$	$23n\tau\lvert\mathbb{G}\rvert$	$36n(l-\tau)\lvert\mathbb{G}\rvert$	$(53nl - 13n\tau)\lvert\mathbb{G}\rvert$

Table 3.2: First-price auction communication complexity comparison in terms of transmitted bits by a party $\mathcal{P}_i \in \mathcal{P}$: n is the number of parties, l is the total number of rounds in Stages 2 and 3 (*i.e.*, the bit-length of bids), τ is the number of rounds of Stage 2, $\lvert\mathbb{G}\rvert$ and $\lvert\mathbb{Z}_q\rvert$ indicate the bit-length of elements $g \in \mathbb{G}$ and $z \in \mathbb{Z}_q$ respectively, λ is the security parameter, as defined in Section 2.

3.8 Complexity analysis and comparison to other protocols

In this section, we present concrete estimates for the computational and communication complexity of our first and second-price auction protocols, *i.e.*, Π_{FPA} and Π_{SPA}, respectively. We show that, in the first-price case, Π_{FPA} is more efficient than the state-of-the-art protocol SEAL [13]. In the second-price case, we show that Π_{SPA} only incurs a small overhead (dominated by re-executing one round) over Π_{FPA}.

The First-Price Case:

A concrete estimate of computational complexity is shown in Table 3.1 and one for communication complexity is shown in Table 3.2. We estimate these concrete complexities in terms of the number of exponentiations performed by a party $\mathcal{P}_i$ and of the number of bits transmitted by a party $\mathcal{P}_i$ in an execution of protocol Π_{FPA}, respectively. Moreover, we compare the complexity of our protocol with SEAL [13], which is the current state-of-the-art protocol for first-price sealed-bid auctions. In a similar way to our protocol, SEAL requires all parties to jointly compute the maximum bid bit-by-bit and is subdivided into a Stage 1 devoted to the setup, a Stage 2 identifying the rounds of the protocol before the first veto and a Stage 3 identifying the rounds of the protocol after the first veto. Hence, we highlight the differences in terms of complexity stage

by stage. Note that, in order to make the communication complexities of the two protocols comparable, both of them have been expressed in terms of $|\mathbb{G}|$. Finally, FAST has an additional Stage 4 guaranteeing that the payment from the winning party $\mathcal{P}_w$ to the auctioneer is executed. On the other hand, SEAL does not guarantee this property. In particular, Stage 4 requires 1 exponentiation per party and has a communication complexity equal to $2(n-1)|\mathbb{G}|$.

The Second-Price Case:

The computational and communication complexities of the proposed second-price auction are still linear in the number of agents. That is, assuming that at round r, there is a party who is the only one that is veto-ing, then the parties have to re-run the r^{th} round with one less party. More precisely, by following the notation of Table 3.1 and 3.2, let τ be the number of rounds in Stage 2, then the computational complexity of Stage 1 and Stage 2 is similar to the first-price auction, that is $nl + l + 8\log l + 2$ for Stage 1, and $8\tau + 10n\tau$ for Stage 2. Let r, be the number of rounds until there is only a single party who is veto-ing. Therefore the computational complexity of Stage 3 is $19r + 22nr$ until there is only a single veto. After this the parties have to run the protocol with one less party, *i.e.*, $n-1$ parties. Depending on the bid structure of the remaining $n-1$ parties, the protocol is either in Stage 2 or Stage 3. Let τ' denote the number of rounds until the remaining $n-1$ parties get a veto. Then the computational complexity for these τ' rounds would be $8\tau' + 10(n-1)\tau'$, and for the remaining $l-(\tau+\tau'+r)$ it would be $19\big(l-(\tau+\tau'+r)\big) + 22(n-1)\big(l-(\tau+\tau'+r)\big)$. Using the same notation, a similar argument follows for the communication complexity per party in the case of the second-price auction.

3.9 Rational strategies

In this section we consider the incentives of parties in our protocols. Note that, the set of bidders is fixed through the execution, *i.e.*, once the execution has started, even if it is required to re-execute the protocol, no new bid can be submitted and it is therefore not possible to gain from the leaked information. Moreover, in case there is a cheating party, the protocols refund the honest parties with her deposit.

We now consider the utility of each party from participating in the protocol. The utility function of a generic party $\mathcal{P}_i$ in the first-price auction is $u_i^{FPA}(b_1, \ldots, b_n) = v_i - b_i$ if $b_i > \max_{j \neq i} b_j$ and 0 otherwise, while in the second-price auction is instead $u_i^{SPA}(b_1, \ldots, b_n) = v_i - \max_{j \neq i} b_j$ if $b_i > \max_{j \neq i} b_j$ and 0 otherwise, where v_i represents the $\mathcal{P}_i$'s private valuation of what is at stake in the auction. It is known that in the first-price auctions the optimal strategy for each rational party depends on their beliefs regarding other party's valuations, while in the second-price auction the optimal strategy for each party is to bid an amount equal to her valuation regardless of the strategy of other parties [131, 142], *i.e.*, $b_i = v_i$.

Note that, in case a party $\mathcal{P}_i$ is honest, she always gets her deposit $work$ back. Then, if she is the winner, she gets what is at stake in the auction and pays b_i, while if she is not the winner, she gets her entire deposit $b_i + work$ back. Therefore, by following the protocol each rational party has a non-negative utility, *i.e.*, $u_i(b_1, \ldots, b_n) \geq 0$. However, if a party cheats her deposit $b_i + work$ is distributed among honest parties. Therefore, the utility of a cheating party, regardless of whether her bid is the highest or not, is $u_i(b_1, \ldots, b_n) = -(b_i + work) < 0$, which is strictly negative. Therefore, cheating is a dominated strategy for each party, *i.e.*, regardless of what other players do it always results in a lower utility.

The above analysis shows that it is not rational for an adversary $\mathcal{A}$ controlling a single party to deviate from the protocol. Next, we show that it is also the case for an adversary $\mathcal{A}$ controlling more than one party. Let $\mathcal{P}_i, \mathcal{P}_j$ be two parties controlled by $\mathcal{A}$ and let $v_{\mathcal{A}}$ be the valuation of the adversary for what is at stake in the auction. Without loss of generality let $b_i > b_j$. If $\mathcal{A}$ does not deviate from the protocol, then her utility is either 0 (in case neither b_i nor b_j is the winning bid) or $v_{\mathcal{A}} - b_i$ (in case b_i is the winning bid). Instead, if $\mathcal{A}$ deviates from the protocol by making $\mathcal{P}_i$ dropout, in case b_j is not the second-highest bid, then her utility is $-(work + b_i)$. If b_j is the second-highest bid, $\mathcal{A}$ gets what is at stake in the auction but her utility is $v_{\mathcal{A}} - (b_i + work + b_j)$. Therefore $\mathcal{A}$ always prefers to behave honestly.

Note that, *it is necessary to have the deposit amount at least equal to the bid.* Indeed, let d be any deposit amount smaller than b_i. Then the utility of $\mathcal{A}$ by making $\mathcal{P}_i$ drop out the protocol is $v_{\mathcal{A}} - (d + work + b_j)$, while it is $v_{\mathcal{A}} - b_i$ by behaving honestly. Therefore, in case $d + work + b_j < b_i$, $\mathcal{A}$ prefers to deviate from the protocol to increase her utility. A similar argument shows that in the second-price auction $\mathcal{A}$ always prefers to act honestly.

Chapter 4

SoK: Mitigation of Front-running in Decentralized Finance

In this Chapter, we present the results that will appear in Financial Cryptography and Data Security, FC 2022 International Workshops, DeFi'22 and are available on Cryptology ePrint Archive [22].

In decentralized finance, we refer to front-running as the malicious act of both manipulating the order of pending trades and injecting additional trades to make a profit at the cost of other users. Given the financial loss and increased transaction load resulting from adversarial front-running, novel cryptographic protocols have been proposed to mitigate such attacks. Thus, we describe common front-running attacks (Section 4.1), propose a schema of front-running mitigation categories (Section 4.2), assess the state-of-the-art techniques in each category and illustrate remaining attacks.

4.1 Front-running attacks

AMM sandwich: We briefly summarize the functionality of constant product AMM's, namely, a liquidity pool holding token balances, r_0 and r_1, of two different token types, τ_0 and τ_1 respectively, s.t. $r_0 \cdot r_1$ is *always* constant when swaps are being carried out between τ_0 and τ_1. A user swaps units of τ_0 for units of τ_1 by authorizing a *left swap* action $\mathsf{SL}(v : \tau_0, w : \tau_1)$. Here, the user is sending $v : \tau_0$ to the AMM in return for at least $w : \tau_1$ (swap limit). For this left swap to be valid, the product of the reserves must be maintained. Thus, the following relation between initial and updated reserves must hold: $r_0 \cdot r_1 = (r_0 + v) \cdot (r_1 - w')$, where $w' \geq w$ and w' represents the units of τ_1 that the user actually gets. We refer w as the swap *limit*. A *right swap* of $\mathsf{SR}(v : \tau_0, w : \tau_1)$ follows similarly: the user sends $w : \tau_1$ for at least $v : \tau_0$ in

return such that $r_0 \cdot r_1 = (r_0 - v') \cdot (r_1 + w)$ and $v' \geq v$ where v' represents the units of τ_0 received. Constant product AMM's exhibit *slippage*: subsequent swaps in the same direction exhibit decreasing exchange rates.

User swaps can be "sandwiched", exploiting slippage for the gain of the attacker. Consider a left swap $\mathsf{A} : \mathsf{SL}(\mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}_\mathsf{A} : \tau_1)$ submitted by user A. A front-run swap by attacker M in the same direction reduces the exchange rate for the subsequent victim swap: a final back-run swap by M in the opposing direction then profits from an improved exchange rate.

$$\mathsf{M} : \mathsf{SL}(\mathsf{v}^\mathsf{f}_\mathsf{M} : \tau_0, \mathsf{w}^\mathsf{f}_\mathsf{M} : \tau_1) \quad \mathsf{A} : \mathsf{SL}(\mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}_\mathsf{A} : \tau_1) \quad \mathsf{M} : \mathsf{SR}(\mathsf{v}^\mathsf{b}_\mathsf{M} : \tau_0, \mathsf{w}^\mathsf{b}_\mathsf{M} : \tau_1)$$

Optimal front-run $(\mathsf{v}^\mathsf{f}_\mathsf{M}, \mathsf{w}^\mathsf{f}_\mathsf{M})$ and back-run $(\mathsf{v}^\mathsf{b}_\mathsf{M}, \mathsf{w}^\mathsf{b}_\mathsf{M})$ parameters are a function of the victim's swap, inferred from the pending victim transaction gossiped across the network [16].

We illustrate a step-wise execution of a sandwich in Figure 4.1 and introduce notation for user and AMM state proposed in [15] for this purpose. The wallet of A is modelled as the term $\mathsf{A}[v_i : \tau_0, ..., v_n : \tau_n]$, where $v_0, ..., v_n$ are the respective balances of token types $\tau_0, ..., \tau_n$. The state of an AMM holding token types τ_0 and τ_1 is given by its reserve balances $(r_0 : \tau_0, r_1 : \tau_1)$. Thus, we express the system state as a composition of wallets and reserve balances.

$$\mathsf{A}[v : \tau] \mid (r_0 : \tau_0, r_1 : \tau_1)$$

Let the initial AMM balance be $(100 : \tau_0, 100 : \tau_1)$. User A wishes to perform the swap $\mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1)$. For simplicity, we assume unit values of τ_0 and τ_1 to be equal: given the ratio of AMM reserves is 1, there is no arbitrage opportunity to be exploited [15]. If A's order is executed immediately, A receives $13 : \tau_1$ for the $15 : \tau_0$ it sends to the AMM. Instead, however, if the user swap is sandwiched by attacker M (Figure 4.1), A only obtains the minimum amount $10 : \tau_1$, implying a reduction of $3 : \tau_1$. Note that the reserve product

$$\begin{aligned}
&\mathsf{A}[15 : \tau_0] \mid \mathsf{M}[15 : \tau_0, 10 : \tau_1] \mid (100 : \tau_0, 100 : \tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(15:\tau_0,13:\tau_1)} &\mathsf{A}[15 : \tau_0] \mid \mathsf{M}[23 : \tau_1] \mid (115 : \tau_0, 87 : \tau_1) \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} &\mathsf{A}[10 : \tau_1] \mid \mathsf{M}[23 : \tau_1] \mid (130 : \tau_0, 77 : \tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(30:\tau_0,23:\tau_1)} &\mathsf{A}[10 : \tau_1] \mid \mathsf{M}[30 : \tau_0] \mid (100 : \tau_0, 100 : \tau_1)
\end{aligned}$$

Figure 4.1: Sandwich attack

is maintained at each execution step and that the sandwich execution preserves the initial reserve ratio: the attack leaves no arbitrage opportunity unexploited. The attacker M's profit of 5 units of τ_0 (or τ_1) is optimal [16]: A receives the minimum amount possible, namely its swap limit.

Scheduled AMM sandwich: For certain AMM variants, the knowledge of the user's intent to perform a swap can be directly inferred from the blockchain state. Paradigm [172] propose scheduled AMM swaps, or more generally, *scheduled inputs*. Let $\mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1, \mathsf{r})$ be a swap that is not executed immediately, but scheduled for evaluation together with the first user-AMM interaction *following* blockchain round r, thus requiring no further interaction from A. Since *scheduled* orders are stored in the AMM smart contract and evaluated at the beginning of a known round, the sandwich attack strategy can be exploited, albeit over two block rounds [172]: the front-run is sequenced at the end of round r and the back-run as the first newly submitted swap of round $\mathsf{r}+1$.

Generalized front-run attacks: In decentralized finance, actions exist which are *profitable* for the authorizing user, but which can also be performed by any other agent with a sufficient balance. In the permissionless blockchain setting, *generalized front-runners*, a term coined by Daian [153], are automated agents that identify profitable, pending transactions, which can be authorized by *any* user, and simply replicate these with their own account, thereby depriving the original transaction submitter of it's profit. Since the security of DeFi applications rely on rational agents to solve for profitable arbitrage [178, 171, 82] and liquidation [158] strategies, the presence of generalized front-running threatens to restrict such opportunities to agents colluding with miners.

4.1.1 Formalization: speculative sandwich

We formalize the example attack trace introduced in Figure 4.3 and prove that the attack strategy is either profitable or cost-neutral for the attacker. Again, we assume unit value of τ_0, τ_1 to be equal, and the initial AMM reserve state to be $(\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$: in this state, there is no arbitrage opportunity to be exploited, simplifying our analysis. We omit both AMM and transaction fees.

The victim A swap direction is *left*, inferred by M from A's public balance of $\mathsf{v}_{\mathsf{A}}^{\mathsf{init}} : \tau_0$ (A holds no units of τ_1). The attack strategy is as follows:

1. **Round r**: Front-run victim with $\mathsf{M} : \mathsf{SL}(\mathsf{v}_{\mathsf{M}}^{\mathsf{f}} : \tau_0, \mathsf{w}_{\mathsf{M}}^{\mathsf{f}} : \tau_1)$ such that
$$(\mathsf{r} + \mathsf{v}_{\mathsf{M}}^{\mathsf{f}}) \cdot (\mathsf{r} - \mathsf{w}_{\mathsf{M}}^{\mathsf{f}}) = \mathsf{r}^2 \tag{4.1}$$
2. **Round r + 1**: Back-run victim in opposing direction to reestablish initial AMM reserve ratio, or if attacker balance is insufficient, back-run with largest amount available to attacker M.

We must show that this strategy is always profitable (when the victim swap direction can be inferred by the attacker). We note that there are several variables beyond the attackers control. The ordering of both front-run and victim swap in round r is random. Thus the desired "front-run" ordering of the victim swap in round r may not succeed (the sandwich is unsuccessful if the victim swap precedes attacker front-run swap). Furthermore, the victim swap parameters can be arbitrarily chosen, so that the victim swap may not be *enabled* or execute in

a given sequence. Thus, we must exhaustively demonstrate the profitability of the attacker strategy for all possible cases:

1) Successful sandwich & enabled victim swap
2) Successful sandwich & disabled victim swap
3) Unsuccessful sandwich & enabled victim swap
4) Unsuccessful sandwich & disabled victim swap

Case 1: *(Successful sandwich & enabled victim swap)*: We illustrate the symbolic execution of the attack trace below in terms of initial balances, chosen swap parameters and exchanged amounts.

	⓪	$\mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)$
		Round r
$\xrightarrow{\mathsf{M}:\mathsf{SL}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0,\mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)}$	①	$\mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1)$
$\xrightarrow{\mathsf{A}:\mathsf{SL}(\mathsf{v}_\mathsf{A}:\tau_0,\mathsf{w}_\mathsf{A}:\tau_1)}$	②	$\mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} - \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}'_\mathsf{A} : \tau_1] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1] \mid$ $(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}'_\mathsf{A} : \tau_1)$
		Round $\mathsf{r}+1$
$\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_\mathsf{M}^\mathsf{b}:\tau_0,\mathsf{w}_\mathsf{M}^\mathsf{b}:\tau_1)}$	③	$\mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} - \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}'_\mathsf{A} : \tau_1] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}_\mathsf{M}^\mathsf{b} : \tau_1] \mid$ $(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} - \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}'_\mathsf{A} + \mathsf{w}_\mathsf{M}^\mathsf{b} : \tau_1)$

We show that the attack is profitable. For τ_0 and τ_1 of equal unit value, the net change in *value* exchanged by M must be positive. Thus, we must prove

$$\text{profit}_\mathsf{M} = -\mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}_\mathsf{M}^\mathsf{b} + \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} > 0 \tag{4.2}$$

Note that the amounts exchanged in the front-run are equal to the front-run parameters $(\mathsf{v}_\mathsf{M}^\mathsf{f}, \mathsf{w}_\mathsf{M}^\mathsf{f})$, as they are chosen such that (4.1) holds. We consider the **sub-case (a)** in which the attacker M has sufficient balance to perform the back-run swap such that the AMM reserves are restored to the original state and the **sub-case (b)** in which the attacker initially has no balance of τ_1 to perform the back-run: $\mathsf{w}_\mathsf{M}^{\mathsf{init}} = 0$. Here, the funds of τ_1 required to execute the back-run are received entirely in the front-run execution.

For **sub-case (a)**, we rewrite (4.2) in terms of independently chosen parameters $\mathsf{v}_\mathsf{M}^\mathsf{f}$, v_A (the attacker only knows the victim swap direction) and initial reserve amounts r. The reserves of the AMM are restored to the initial state in final state ③: summing all step changes to the reserves across the sandwich execution yields

$$\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} - \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} = \mathsf{r} \qquad \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}'_\mathsf{A} + \mathsf{w}_\mathsf{M}^\mathsf{b} = \mathsf{r}$$
$$\mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} - \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} = 0 \qquad -\mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}'_\mathsf{A} + \mathsf{w}_\mathsf{M}^\mathsf{b} = 0$$

or

$$v_M^{b\,\prime} = v_M^f + v_A \qquad w_M^b = w_M^f + w'_A$$

Inserting RHS of equations above into our proof obligation (4.2) yields

$$\begin{aligned} \text{profit}_M = -\cancel{v_M^f} + \cancel{v_M^f} + v_A + \cancel{w_M^f} - \cancel{w_M^f} - w'_A >^? 0 \\ v_A - w'_A >^? 0 \end{aligned} \tag{4.3}$$

To evaluate whether this inequality holds, we must solve for w'_A in terms of v_A and v_M^f chosen independently by the victim and adversary respectively. We exploit the constant reserve product invariant which holds for across the entire execution.

$$\begin{aligned} (r + v_M^f) \cdot (r - w_M^f) &= r^2 \quad \text{(front-run swap)} \\ (r + v_M^f + v_A) \cdot (r - w_M^f - w'_A) &= r^2 \quad \text{(victim swap)} \end{aligned}$$

We can derive $r - w_M^f = \frac{r^2}{r+v_M^f}$ from the first equation, and substitute the RHS for $r - w_M^f$ in the second equation to obtain

$$(r + v_M^f + v_A) \cdot (\frac{r^2}{r + v_M^f} - w'_A) = r^2$$

Solving for w'_A ...

$$\begin{aligned} w'_A &= \frac{r^2}{r + v_M^f} - \frac{r^2}{r + v_M^f + v_A} \\ &= \frac{r^2(r + v_M^f + v_A) - r^2(r + v_M^f)}{(r + v_M^f)(r + v_M^f + v_A)} \\ &= \frac{r^2}{r^2 + (2v_M^f + v_A)r + (v_M^f)^2 + v_A v_M^f} \cdot v_A \end{aligned}$$

and substituting the RHS for w'_A in the proof obligation in (4.3) finally yields

$$\text{profit}_M = (1 - \frac{r^2}{r^2 + (2v_M^f + v_A)r + (v_M^f)^2 + v_A v_M^f}) \cdot v_A > 0 \tag{4.4}$$

The fraction expression above is less than 1 for any choice of positive v_M^f and v_A as the numerator is smaller than the denominator. The attacker profit is thus positive and increases with v_M, justifying the front-run swap by M.

Next, we consider the **sub-case (b)**, where the attacker initially has no balance of τ_1, and restate the profit of attacker for the reader's convenience.

$$\text{profit}_M = -v_M^f + w_M^f - w_M^b + v_M^{b\,\prime} >^? 0$$

We assume initial attacker balance in $\mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1$ to be $0 : \tau_1$, so that all the amount of τ_1 available for the back-run in state ② is received in the front-run: thus, substituting $\mathsf{w}_\mathsf{M}^\mathsf{b} = \mathsf{w}_\mathsf{M}^\mathsf{f}$ into the equation above yields

$$\mathsf{profit}_\mathsf{M} = -\mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} >^? 0 \tag{4.5}$$

To prove this inequality, we solve for $\mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime}$ in terms of $\mathsf{v}_\mathsf{M}^\mathsf{f}$ and v_A chosen independently by the victim and adversary respectively and initial reserves amounts r. We exploit the constant reserve product invariant which holds throughout the execution.

$$\begin{aligned}
(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}) \cdot (\mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f}) &= \mathsf{r}^2 \quad \text{(Front-run)} \\
(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}) \cdot (\mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}_\mathsf{A}') &= \mathsf{r}^2 \quad \text{(Victim swap)} \\
(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} - \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime}) \cdot (\mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \mathsf{w}_\mathsf{A}' + \mathsf{w}_\mathsf{M}^\mathsf{b}) &= \mathsf{r}^2 \quad \text{(Back-run)}
\end{aligned}$$

Since $\mathsf{w}_\mathsf{M}^\mathsf{f} = \mathsf{w}_\mathsf{M}^\mathsf{b}$ is assumed in sub-case (b), the 3rd equation (back-run) yields

$$\mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} = \mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A} - \frac{\mathsf{r}^2}{\mathsf{r} - \mathsf{w}_\mathsf{A}'} \tag{4.6}$$

From the 2nd equation (victim swap), we solve for w_A' in terms of independent parameters $\mathsf{v}_\mathsf{M}^\mathsf{f}$, v_A and r

$$\mathsf{w}_\mathsf{A}' = \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}}$$

From the 1st equation (front-run) $\mathsf{w}_\mathsf{M}^\mathsf{f} = \frac{\mathsf{r} \cdot \mathsf{v}_\mathsf{M}^\mathsf{f}}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}}$, so we can rewrite the above as

$$\mathsf{w}_\mathsf{A}' = \mathsf{r} - \frac{\mathsf{r} \cdot \mathsf{v}_\mathsf{M}^\mathsf{f}}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}} = \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}} - \frac{\mathsf{r}^2}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}} = \frac{\mathsf{r}^2 \cdot \mathsf{v}_\mathsf{A}}{(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A})}$$

$$\mathsf{r} - \mathsf{w}_\mathsf{A}' = \frac{\mathsf{r}(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}) - \mathsf{r}^2 \cdot \mathsf{v}_\mathsf{A}}{(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A})}$$

Substituting the RHS above for $\mathsf{r} - \mathsf{w}_\mathsf{A}'$ in the denominator expression of (4.6) and then substituting the RHS of (4.6) for $\mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime}$ in (4.5) yields

$$\begin{aligned}
\mathsf{profit}_\mathsf{M} &= -\cancel{\mathsf{v}_\mathsf{M}^\mathsf{f}} + \mathsf{r} + \cancel{\mathsf{v}_\mathsf{M}^\mathsf{f}} + \mathsf{v}_\mathsf{A} - \frac{\mathsf{r}^2(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A})}{\mathsf{r}(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}) - \mathsf{r}^2 \cdot \mathsf{v}_\mathsf{A}} \\
&= \mathsf{v}_\mathsf{A} - \frac{\mathsf{r}^3 \mathsf{v}_\mathsf{A}}{\mathsf{r}(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}) - \mathsf{r}^2 \cdot \mathsf{v}_\mathsf{A}} \\
&= (1 - \frac{\mathsf{r}^2 \mathsf{v}_\mathsf{A}}{(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f})(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} + \mathsf{v}_\mathsf{A}) - \mathsf{r} \cdot \mathsf{v}_\mathsf{A}}) \cdot \mathsf{v}_\mathsf{A} \\
&= (1 - \frac{\mathsf{r}^2}{\mathsf{r}^2 + 2\mathsf{v}_\mathsf{M}^\mathsf{f}\mathsf{r} + (\mathsf{v}_\mathsf{M}^\mathsf{f})^2 + \mathsf{v}_\mathsf{A}\mathsf{v}_\mathsf{M}^\mathsf{f}}) \cdot \mathsf{v}_\mathsf{A}
\end{aligned} \tag{4.7}$$

The attacker profit is positive but strictly less than the gain (4.4) obtained in sub-case (a).

Case 2 *(Successful sandwich & disabled victim swap)*: Should the victim swap not execute in round r, then M can simply revert the state of the AMM with a back-run in the round $r+1$ with the same parameter values as in the front-run.

$$\begin{array}{rl}
 & \textcircled{0} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{A}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0, \mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)} & \textcircled{1} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1) \\
\xrightarrow{\mathsf{A}:\cancel{\mathsf{SL}(\mathsf{v}_\mathsf{A}:\tau_0, \mathsf{w}_\mathsf{A}:\tau_1)}} & \textcircled{2} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0, \mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)} & \textcircled{3} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)
\end{array}$$

The attack execution is trivially cost-neutral for M.

Case 3 *(Failed sandwich & enabled victim swap)*: We must show that the attacker front-run must be disabled assuming the attacker parameters are chosen as described in the attack strategy. Further, we can demonstrate that the back-run by the attacker is profitable.

$$\begin{array}{rl}
 & \textcircled{0} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(\mathsf{v}_\mathsf{A}:\tau_0, \mathsf{w}_\mathsf{A}:\tau_1)} & \textcircled{1} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} - \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}'_\mathsf{A} : \tau_1] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{r} - \mathsf{w}'_\mathsf{A} : \tau_1) \\
\xrightarrow{\mathsf{M}:\cancel{\mathsf{SL}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0, \mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)}} & \textcircled{2} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} - \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}'_\mathsf{A} : \tau_1] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{r} - \mathsf{w}'_\mathsf{A} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_\mathsf{M}^\mathsf{b}:\tau_0, \mathsf{w}_\mathsf{M}^\mathsf{b}:\tau_1)} & \textcircled{3} \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} - \mathsf{v}_\mathsf{A} : \tau_0, \mathsf{w}'_\mathsf{A} : \tau_1] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} + \mathsf{v}_\mathsf{M}^{\mathsf{b}\,\prime} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} - \mathsf{w}_\mathsf{M}^\mathsf{b} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)
\end{array}$$

As described in step (1) of attack strategy, M's front-run parameters are chosen such that

$$\begin{aligned}
(\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}) \cdot (\mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f}) &= \mathsf{r}^2 \\
\mathsf{w}_\mathsf{M}^\mathsf{f} &= \frac{\mathsf{r} \cdot \mathsf{v}_\mathsf{M}^\mathsf{f}}{\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f}}
\end{aligned} \tag{4.8}$$

Thus, the front-run swap is only enabled if the received amount is equal or greater to $\mathsf{w}_\mathsf{M}^\mathsf{f}$ shown above. Note, that this doesn't hold if the front-run is executed in state ① of case (3) following the enabled victim swap. We prove this by contradiction: assume that the front-run executes following the victim swap, then the constant reserve product invariant must hold.

$$\begin{aligned}
(\mathsf{r} + \mathsf{v}_\mathsf{A}) \cdot (\mathsf{r} - \mathsf{w}'_\mathsf{A}) &= \mathsf{r}^2 \quad \text{(Victim swap)} \\
(\mathsf{r} + \mathsf{v}_\mathsf{A} + \mathsf{v}_\mathsf{M}^\mathsf{f}) \cdot (\mathsf{r} - \mathsf{w}'_\mathsf{A} - \mathsf{w}_\mathsf{M}^{\mathsf{f}\,\prime}) &= \mathsf{r}^2 \quad \text{(Front-run)}
\end{aligned}$$

We solve for $(r - w'_A)$ in the first equation and insert into the second equation to obtain

$$(r + v_A + v_M^f) \cdot (\frac{r^2}{r + v_A} - w_M^{f\,\prime}) = r^2$$

Further, we solve for $w_M^{f\,\prime}$ in terms of r, v_A and v_M^f

$$\frac{r^2}{r + v_A} - w_M^{f\,\prime} = \frac{r^2}{(r + v_A + v_M^f)}$$

$$w_M^{f\,\prime} = \frac{r^2}{r + v_A} - \frac{r^2}{r + v_A + v_M^f} = \frac{r^2 \cdot v_M^f}{(r + v_A) \cdot (r + v_A + v_M^f)} = \frac{r}{r + v_A} \cdot \frac{r \cdot v_M^f}{(r + v_A + v_M^f)}$$

Comparing with w_M^f in (4.8), we can infer the following inequality

$$w_M^{f\,\prime} < w_M^f$$

which cannot hold in a valid execution by definition of swaps: a user cannot receive less than the chosen swap limit. Thus, the front-run cannot be enabled in state ① of case (3).

Next, we prove the profitability of the back-run. Assuming a sufficient balance of the attacker to revert the effect of the victim swap, the swap parameters of the back-run can be chosen to reverse the affects of victim swap on the AMM reserves, which M observes following the output-phase of round r: namely, $v_M^b = v_A$ and $w_M^b = w_A{}'$. We insert these into the reserve product invariant from the victim swap

$$(r + v_A) \cdot (r - w_A{}') = r^2 \quad \text{(Victim swap)}$$

to obtain

$$\begin{aligned} (r + v_M^b) \cdot (r - w_M^b) &= r^2 \\ w_M^b &= \frac{r}{r + v_M^b} \cdot v_M^b \\ w_M^b &< v_M^b \end{aligned}$$

For equal unit value of both token types, this is clearly profitable, as M receives more value (v_M^b) as it sends (w_M^b). If attacker has no balance of τ_1 it simply omits the back-run and the attack is aborted, resulting in a cost-neutral execution for the attacker.

Case 4 *(Failed sandwich & disabled victim swap)*: As in case (2) - should the victim swap not execute in round r, then M can simply revert the state of the AMM with a back-run in the round $r + 1$

$$\begin{array}{rl}
 & ⓪ \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{A}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(\mathsf{v}_\mathsf{A}:\tau_0,\mathsf{w}_\mathsf{A}:\tau_1)} & ① \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1) \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0,\mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)} & ② \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} - \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} + \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1] \mid (\mathsf{r} + \mathsf{v}_\mathsf{M}^\mathsf{f} : \tau_0, \mathsf{r} - \mathsf{w}_\mathsf{M}^\mathsf{f} : \tau_1) \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(\mathsf{v}_\mathsf{M}^\mathsf{f}:\tau_0,\mathsf{w}_\mathsf{M}^\mathsf{f}:\tau_1)} & ③ \quad \mathsf{A}[\mathsf{v}_\mathsf{A}^{\mathsf{init}} : \tau_0] \mid \mathsf{M}[\mathsf{v}_\mathsf{M}^{\mathsf{init}} : \tau_0, \mathsf{w}_\mathsf{M}^{\mathsf{init}} : \tau_1] \mid (\mathsf{r} : \tau_0, \mathsf{r} : \tau_1)
\end{array}$$

The attack execution is trivially cost-neutral for M.

4.1.2 Speculative sandwich with private user balances

Importantly, when performing the speculative AMM swap attack as shown in 4.1.1, the direction of the victim swap must be known. If user balances are private, M will have to guess the direction of the front-running swap. However, this is not a profitable strategy: an incorrect guess can result in a loss for M as shown in the trivial example execution below.

$$\begin{array}{rl}
 & \mathsf{A}[10 : \tau_0, 10 : \tau_1] \mid \mathsf{M}[7 : \tau_0, 15 : \tau_1] \mid (100 : \tau_0, 100 : \tau_1) \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(7:\tau_0,6.5:\tau_1)} & \mathsf{A}[10 : \tau_0, 10 : \tau_1] \mid \mathsf{M}[21.5 : \tau_1] \mid (107 : \tau_0, 93.5 : \tau_1) \\
\xrightarrow{\mathsf{A}:\mathsf{SR}(17:\tau_0,6.5:\tau_1)} & \mathsf{A}[7 : \tau_0, 3.5 : \tau_1] \mid \mathsf{M}[21.5 : \tau_1] \mid (100 : \tau_0, 100 : \tau_1)
\end{array}$$

Again, assuming equal unit value of τ_0 and τ_1, M realizes a loss of $7+15-21.5 = 0.5$. No back-run swap is possible that extracts any arbitrage value given that the reserve ratio is already consistent with the assumption that unit values of τ_0 and τ_1 are equal [15]. Thus, speculative sandwich attacks are only rational if the victim swap direction can be inferred, motivating the need for private user balances.

4.1.3 Example: speculative sandwich of scheduled swap

We illustrate an example of a sandwich of a scheduled swap. Such an attack can be exploited despite the batching of blinded user inputs Section 4.2.2, as long as input schedules remain public. Let $\mathsf{A} : \mathsf{SL}(20 : \tau_0, 15 : \tau_1, \mathsf{r})$ be a swap action that is scheduled to execute as soon as possible *following* block-chain round r, thus requiring no further interaction from the user. Further, let the set of scheduled swap orders be captured in a publicly observable state fragment, *i.e.*, $\Gamma = [\,\mathsf{A} : \mathsf{SL}(15 : \tau_0, 10 : \tau_1, \mathsf{r})\,]$. In practice, such a scheduled swap order will be *evaluated* prior to the first swap order in round $\mathsf{r}+1$, so that it is not possible for the adversary to place a front-run swap before it in round $\mathsf{r}+1$.

However, the sandwich attack can still be executed by an adversary which prevents honest users from submitting swap. The adversary simply submits the

front-run to round r, and the back-run to round $\mathsf{r}+1$, whilst suppressing all other user inputs.

$$\begin{array}{rl}
 & \mathsf{A}[15:\tau_0] \mid \mathsf{M}[15:\tau_0, 10:\tau_1] \mid (100:\tau_0, 100:\tau_1) \mid \Gamma \\
\hline
 & \text{Round } \mathsf{r} \\
\xrightarrow{\mathsf{M}:\mathsf{SL}(15:\tau_0,13:\tau_1)} & \mathsf{A}[15:\tau_0] \mid \mathsf{M}[23:\tau_1] \mid (115:\tau_0, 87:\tau_1) \mid \Gamma \\
\hline
 & \text{Round } \mathsf{r}+1 \\
\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1,\mathsf{r})} & \mathsf{A}[10:\tau_1] \mid \mathsf{M}[23:\tau_1] \mid (130:\tau_0, 77:\tau_1) \mid \\
 & \Gamma \setminus [\,\mathsf{A}:\mathsf{SL}(15:\tau_0, 10:\tau_1), \mathsf{r}\,] \\
\xrightarrow{\mathsf{M}:\mathsf{SR}(30:\tau_0,23:\tau_1)} & \mathsf{A}[10:\tau_1] \mid \mathsf{M}[30:\tau_0] \mid (100:\tau_0, 100:\tau_1) \mid \\
 & \Gamma \setminus [\,\mathsf{A}:\mathsf{SL}(15:\tau_0, 10:\tau_1, \mathsf{r})\,]
\end{array}$$

We emphasize that scheduled swap orders do not require the submitting user A to participate in the round it is scheduled: it is evaluated automatically by the application. Furthermore, since the victim's swap parameters are public, the front-run and back-run parameters can be chosen to optimize M's profit.

4.1.4 Speculative sandwich in hash-based commit & reveal schemes

As shown in Section 4.1.1, the speculative sandwich attack is rational as long as the direction of the victim swap is known. Hash-based commit & reveal schemes suffer from selective output by the adversary (Figure 4.2), permitting a speculative attack to succeed even if the swap direction cannot be inferred from public user balances. Here the attacker simply commits two front-run swaps of opposing directions in the same round as the victim swap, whilst suppressing other user inputs. In the output-phase, the adversary learns the direction of the victim swap before having to open its own commitments and selectively opens the front-run of the same direction as the victim swap, whilst refraining from opening the other front-run swap. The back-run is then executed as in Section 4.1.1.

4.2 Mitigation categories

4.2.1 Fair ordering

A recent line of research [136, 123, 124] has formalized an intuitive notion of γ-*receipt-order-fairness*: given two distinct transactions tx and tx' broadcast by users, receipt-order-fairness of a consensus protocol ensures that tx will be finalized prior to tx' if a γ fraction of network nodes receives tx prior to tx'. However, Kelkar et al. [123] show that even if all nodes agree on the relative order in which any *pair* of transactions were first observed at the gossip stage,

a global transaction ordering of all transactions consistent with the local view of pair-wise orderings is not always possible (Condorcet Paradox). Instead, a weaker notion of γ-*batch-order-fairness* is realized in [124], where tx will be sequenced prior to or in the same block as tx′ if a γ node fraction receives tx first.

Front-running despite fair ordering: Although order fairness removes the miner or round leader's privilege to sequence transactions, it assumes that users have secure channels to servers participating in consensus: in practice, however, public blockchains rely on gossip networks to propagate pending transactions. Here, the *rushing* network adversary can control the receipt-order of transactions for each consensus node, thereby rendering the notion of γ-*batch-order-fairness* meaningless. In practice, such a network adversary model may be excessively strong: whereas in the standard setting the miner or round leader incurs no additional cost for front-running victims, a non-trivial communication cost is now imposed on the rushing adversary. Still, since order-fairness clearly cannot *eliminate* front-running attacks in the (realistic) gossip-network setting, the motivation for stronger front-running mitigation properties remains.

4.2.2 Batching of blinded inputs

Batching of blinded inputs is a technique to ensure 1) the independence between user inputs and 2) the prevention of any adversarial sequencing of inputs. Interactions occur in rounds: in each, inputs are committed during the *input-phase*, followed by an *output phase* where the application state is updated after evaluating user inputs with valid parameters. The *collection* of inputs can occur in a smart contract or by a committee executing a cryptographic protocol which authorizes the distribution of funds from a smart contract in the output phase. The update of the application state following each round can result from the evaluation of valid inputs in *randomized* order or an application-specific *aggregation* thereof: for example, a subset of submitted AMM swaps can be aggregated into a single resulting swap.

		Input independence	Input privacy	Open challenges
Commit & reveal	Hash commitments*	-	-	*Output bias*
	Timed commitments*	•	-	*Delay parameters*
	Threshold encryption**	•	-	*Honest majority*
	Secure multi-party computation**	•	-	*Honest majority*
Input aggregation		•	•	*Abort penalty*
	Homomorphic encryption**	•	•	*Efficiency*

Figure 4.2: Batching of blinded inputs sent to a smart contract* or committee**

In batching of blinded inputs, we distinguish between **commit & reveal** and

input aggregation (Figure 4.2). Both schemes commit inputs in the input-phase of each round, thereby ensuring input independence. However, while input aggregation keeps the users' input private indefinitely, commit & reveal schemes leak individual user inputs when commitments are opened, thereby offering no *input privacy* by definition. Input privacy is necessary to prevent front-running in *subsequent* interaction rounds: past inputs leak information about updates to private balances (Section 4.2.3), which in turn can be exploited by front-runners, as balances constrain the valid user input space.

$$\text{Past user inputs} \xrightarrow{\text{reveal}} \text{Private user balances} \xrightarrow{\text{reveal}} \text{Future user inputs}$$

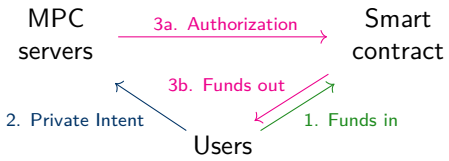
In contrast, input aggregation only outputs the application state update: for aggregated AMM swaps, only reserve updates are revealed, and updates to user balances remain private, if private balances are supported. Naturally, input aggregation can only offer input privacy up to the input batch size.

Commit & reveal: Although *hash commitments* collected by a smart contract may appear to be an obvious approach to implement the commit & reveal functionality, they suffer from *output bias*, as the adversary can selectively refrain from opening its commitment.

Time-lock puzzles [160] or *timed commitments* [39] generated by users and sent to a smart contract promise to eliminate output bias, since the adversary's commitment can be force-opened after a delay, guaranteeing the inclusion of its input in the output-phase. However, in the worst case, each user time-locked input must be solved separately by a constant number of squaring operations in a randomly sampled group, potentially rendering the approach impractical for larger batches of time-locked inputs [125]. Burdges and De Feo [50] propose a novel *delay encryption* notion and construction, which promises encryption of many inputs to a randomly sampled *session key*. Thus, all delay-encrypted inputs of a given batch can be decrypted after a single extraction process. Delay encryption [50] is constructed from isogeny-based cryptography, a recent and less-well studied class cryptographic assumptions. Finally, it remains an open challenge to match delay cryptography parameters to real-world delays which depend on assumed gate speeds used in practice.

Threshold encryption [89] can realize a commit & reveal scheme with the assumption of an *honest majority* committee holding trapdoor information of the encrypted inputs [164]. In each round, a key pair is produced by the execution of a *distributed key generation* (DKG) protocol and the public is opened, with which users encrypt their inputs in the given round. A subsequent opening of the corresponding secret key by the threshold committee enables the decryption of all inputs of the given round. However, should an encrypted user input fail to be finalized in the block-chain in a given round due to network congestion, the user's intent will be made public after the secret key is revealed for the given round without the user action being executed. Given this leakage, the front-running adversary may now anticipate the re-submission of the same user input in the next round.

Secure multi-party computation [175, 107] (MPC) has been proposed [140, 6] to realize a commit & reveal functionality with guaranteed input reveal in an anonymous fashion, also formalized as *anonymous committed broadcast* (ACB) in [6]. The anonymization of inputs is achieved by random *shuffling* of user inputs in an efficient manner. Here, *honest majority* MPC protocols [24, 80] are favoured, as the output is guaranteed as long as the honest majority assumption holds true. To implement a DeFi application with MPC, an MPC-controlled smart contract is required, to which users send their funds prior to each round.



In the output phase of each MPC round, funds in the smart contract are redistributed to users according to the output(s) of the MPC execution. In practice, users can safely delegate the MPC execution to a group of servers [6].

Input aggregation: Naturally, MPC can realize any aggregation function over private user inputs, and in some instances in an efficient manner. Given the emphasis on the privacy of inputs, *dishonest majority* MPC protocols [62, 29, 81] are favoured, which ensure that private inputs can never be obtained by the adversary as long as a single participant remains honest. Informal proposals to implement AMM instances in a dishonest majority MPC have been proposed by Li et al. [137]. Although dishonest majority MPC can be aborted by a single dishonest party, a recent line of research [128, 20, 21] has realized an efficient set of protocols that identify and financially punish the aborting adversary. This achieves a weaker notion of fairness as the rational adversary is incentivized to never abort. Still, the penalty must exceed the financial *option* value of aborting in order to be effective: given that inputs are private, it remains an open research question on how to size financial penalties for identifiable abort in MPC.

Penumbra [157] proposes the use of *homomorpic encryption* to realize the secure aggregation of homomorphically encrypted AMM swap orders. The aggregated swap is then decrypted to reveal the updated AMM reserves. User balances are implemented with private coins (see Section 4.2.3), thus the privacy of the inputs are only dependent on the batch size. We note the non-trivial complexity of aggregating a batch of encrypted AMM swaps with swap limit constraints: *efficient* secure multi-party computation with fully homomorphic encryption schemes remains an open research problem [102]. In [157], consensus validators are proposed to perform the secure computation, consolidating MPC and consensus layers.

Speculative sandwich w/public user balances: We illustrate that batching of blinded inputs alone is not sufficient to prevent front-running attacks. In-

stead, speculative AMM sandwich attacks are possible in blinded input batching schemes as long as the direction of the victim swap is known by the adversary. This can be inferred from *public* user balances, as detailed in the subsequent example. Such speculative sandwich attacks on batched inputs also assume that the adversary in the permissionless setting can "isolate" a single victim's input in a given round, such that only front-run and victim transactions remain: we argue that each batching round has participant limits due to gas constraints or number of clients that MPC servers can support. Thus, the adversary can occupy any arbitrary number of user slots per round and provide invalid inputs[1] on slots not dedicated to the front-running swap.

Round r		Round r+1
$\mathsf{M} : \mathsf{SL}(\mathsf{v}^{\mathsf{f}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{f}}_{\mathsf{M}} : \tau_1)$	$\mathsf{A} : \mathsf{SL}(\mathsf{v}_{\mathsf{A}} : \tau_0, \mathsf{w}_{\mathsf{A}} : \tau_1)$	$\mathsf{M} : \mathsf{SR}(\mathsf{v}^{\mathsf{b}}_{\mathsf{M}} : \tau_0, \mathsf{w}^{\mathsf{b}}_{\mathsf{M}} : \tau_1)$

Figure 4.3: Speculative sandwich

In this speculative attack, we assume that private AMM swaps in each blinded input batch are evaluated in a *random* order, as proposed in [137, 6]. The front-running M can only speculate on achieving the correct order to execute the sandwich. Since balances are public, M can observe that A's balance of τ_1 is zero: thus, A's submitted swap to the AMM (τ_0, τ_1) must be in the *left* direction. M submits the *front-run* swap in the same direction as the victim in the initial round r.

In the optimistic case shown in Figure 4.3, M's front-run swap is evaluated *prior* to the victim swap (in round r), thus enabling M to position the profitable back-run swap in round r+1, where all other users are prevented from submitting inputs. M's front-run parameters can be chosen such that the front-run swap simply does not execute should the front-run *not* be ordered prior to the victim swap in round r, thereby aborting the attack. We refer to Section 4.1.1 for the proof that this speculative sandwich is rational for the attacker.

An execution of a speculative sandwich is shown in Figures 4.4,4.5: here, adversary M observes victim A's interaction with an AMM which batches blinded inputs. A has a public balance of $20 : \tau_0$ only, allowing M to infer that A can only perform a *left* swap from τ_0 to τ_1 with an input amount of at most $20 : \tau_0$. The attack strategy is executed over two subsequent rounds beginning in the initial state shown in Figure 4.4, where we assume unit values of τ_0 and τ_1 are equal.

In the first round r, M submits the *front-run* swap in the same direction as the victim's, with *arbitrarily chosen* input amount $7 : \tau_0$. The minimum output amount or swap limit of the front-run is then is chosen to be $6.5 : \tau_1$ such that $(100 + 7) \cdot (100 - 6.5) = 100^2$ holds: thus, if the front-run were executed in the initial state, M would receive *exactly* its swap limit. Since all other user orders (other than the victim swap of A) are suppressed, there is a probability of 0.5 that the front-run is randomly evaluated *before* the victim's swap, as shown in

[1] *e.g.*, AMM swap parameters which cannot be executed in the current AMM state.

$$\mathsf{A}[20:\tau_0] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (100:\tau_0, 100:\tau_1)$$

Round r

$$\xrightarrow{\mathsf{M}:\mathsf{SL}(7:\tau_0,6.5:\tau_1)} \mathsf{A}[20:\tau_0] \mid \mathsf{M}[21.5:\tau_1] \mid (107:\tau_0, 93.5:\tau_1)$$

$$\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} \mathsf{A}[5:\tau_0, 11.5:\tau_1] \mid \mathsf{M}[21.5:\tau_1] \mid (122:\tau_0, 82:\tau_1)$$

Round r + 1

$$\xrightarrow{\mathsf{M}:\mathsf{SR}(22:\tau_0,18:\tau_1)} \mathsf{A}[5:\tau_0, 11.5:\tau_1] \mid \mathsf{M}[22:\tau_0, 3.5:\tau_1] \mid (100:\tau_0, 100:\tau_1)$$

Figure 4.4: Successful speculative sandwich

Figure 4.4. The *back-run* swap of M in the opposing direction then follows in the subsequent round with probability 1, since M suppresses all user actions other than its own back-run. Assuming equal unit value of both token types, the attack profit for M is 3.5.

Should the front-run ordering fail (Figure 4.5), then M's front-run parameters are chosen such that the front-run swap will not execute, resulting in an abort of the speculative sandwich attack. This is due to the chosen front-run parameters: following the execution step of A's swap in Figure 4.5, the constant product invariant can only hold if M receives $5:\tau_1$ for the $7:\tau_0$ it sends: $(115+7)\times(87-5)=100^2$. However, this contradicts M swap limit of $6.5:\tau_1$, such that the front-run cannot execute in the state following A's swap. M can still perform a back-run in round r+1, thereby restoring the initial reserve ratio and extracting an arbitrage profit of 2, which is less than in the successful speculative sandwich execution in Figure 4.4. Still, the speculative sandwich attack is always profitable, as shown in Section 4.1.1.

$$\mathsf{A}[20:\tau_0] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (100:\tau_0, 100:\tau_1)$$

Round r

$$\xrightarrow{\mathsf{A}:\mathsf{SL}(15:\tau_0,10:\tau_1)} \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (115:\tau_0, 87:\tau_1)$$

$$\xrightarrow{\mathsf{M}:\mathsf{SL}(7:\tau_0,6.5:\tau_1)} \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[7:\tau_0, 15:\tau_1] \mid (115:\tau_0, 87:\tau_1)$$

Round r + 1

$$\xrightarrow{\mathsf{M}:\mathsf{SR}(15:\tau_0,13:\tau_1)} \mathsf{A}[5:\tau_0, 13:\tau_1] \mid \mathsf{M}[22:\tau_0, 2:\tau_1] \mid (100:\tau_0, 100:\tau_1)$$

Figure 4.5: Aborted speculative sandwich

Importantly, if victim A's swap direction were unknown, M would have to guess the direction of the front-running swap. An incorrect guess can result in a loss for M as shown in Section 4.1.2. Thus, we argue that private user balances are necessary for batching of blinded inputs to be effective. Furthermore, for *scheduled* AMM orders introduced in [172], private user balances remain insufficient if scheduled orders are stored in public smart contracts: we sketch a speculative sandwich attack on publicly scheduled swaps in Section 4.1.3. Finally, we note that hash-based commit & reveal schemes permit speculative

sandwich attacks even when user balances are private, as the adversary can selectively reveal the appropriate sandwich strategy which matches on the swap first revealed by the victim (Section 4.1.4).

4.2.3 Private & secret state

As argued in Section 4.2.2, both the *aggregation* of blinded inputs and use of *private balances and secret input stores* is necessary to mitigate front-running in the current and future rounds. Whilst it may be possible to maintain the *entire* DeFi application state secretly in an MPC instance in order to prevent front-running, this will naturally reduce its utility to users in the permissionless setting. Notably, Angeris et al. [9, 71] argue that both *marginal price* and *validity* of a given AMM swap order must be queryable for an AMM interaction to be meaningful. Therefore, we restrict our study of secret state in DeFi applications to *user input stores* [172, 77], which maintain submitted inputs until they are evaluated or executed at a later point in time.

Private user balances: Private block-chain currencies and tokens have been realized with zero-knowledge proof systems: *confidential transactions* [145] shield output amounts with efficient zero-knowledge *range proofs* [49], thereby ensuring that newly created output values do not exceed those spent by the same transaction. Confidential transactions only shield output amounts: a transaction graph connecting outputs can still be inferred from public transactions on the block-chain, permitting coin taint to propagate downstream.

Z-cash [161] style *decentralized anonymous payment* (DAP) schemes break such public links between outputs, as well-formed relations between new and spent outputs are not revealed but publicly verifiable with SNARK [113, 101, 154, 28, 114] zero-knowledge proofs. DAP schemes have also been proposed for DeFi functionality in Manta [73], but here front-running is not mitigated, since the AMM reserve state is public and swap inputs are not batched. Even though swap parameters are blinded in Manta, each individual swap execution results in a *public* update of AMM reserves. Thus, the *affect* of each swap on the current AMM reserves is known, leaking exchanged amounts and permitting sandwich attack strategies.

Importantly, when implementing input batching (Figure 4.2) with secure computation *and* block-chains supporting private user balances, zero-knowledge proofs must be generated inside the MPC instance in order to update private user balances. Doing so *efficiently* in MPC or even fully homomorphic encryption remains on open research question.

Finally, Submarine commitments [46] propose that users can rely on k-anonymity alone to privately commit funds during the input-phase without the use of private balances. Here, users commit value to an *k-anonymized* address which can only be withdrawn by a specific smart contract after the address is revealed together with the input by the user.

Secret input stores: We note that shielded scheduled AMM swaps [172] or long-running order lists [77] cannot be maintained by encryption alone: encryption of a scheduled swap by a user implies its decryption at a later stage, requiring repeated user interaction, and thus defeating the purpose of scheduled inputs. Alternatively, a decryption by an honest majority committee implies that the round or block-height of the input schedule is known. Instead, we suggest a long-running MPC instance to realize secret input stores in decentralized finance. Here, stored inputs are secret shared across MPC servers: in each round, both newly submitted inputs and secretly stored inputs are secretly evaluated together to update the application state, neither being visible to the front-running adversary.

Chapter 5

PAPR: Publicly Auditable Privacy Revocation for Anonymous Credentials

In this Chapter, we present the results that will appear in CT-RSA 2023, Cryptographers' Track at RSA Conference [47].

We enrich anonymous credential systems by introducing the notion of anonymous credentials with *Publicly Auditable Privacy Revocation* (PAPR), formalize it as an ideal functionality (Section 5.2) and propose a realization that is secure under standard assumptions in the Universal Composability (UC) framework (Section 5.3) against static adversaries. Furthermore, we show how to modify our construction to make it secure against mobile adversaries (Section 5.4).

5.1 Our Techniques

At a high level, our approach to create an anonymous credential scheme with publicly accountable privacy revocation can be summarized in the following three steps. First, the system maintains one global public list of enrolled parties $\mathcal{P}$ (committee candidates), consisting of party identifiers $ID_{\mathcal{P}}$, *e.g.*, a name, and identity keys $\mathsf{pk}_{\mathcal{P}}$ (leveraging a PKI). Second, the issuer produces credentials for a user, only if: *(a)* the user proves to have shared their identity key to an *anonymous* committee, *(b)* the committee is composed by a fixed number of *other parties* in the system (*i.e.*, from the committee candidates), *(c)* the selection of committee parties was *provably at random*. Third, any credential can be subject to privacy revocation upon public announcement. The goal of privacy revocation is to let an authority identify the holder of a given *anonymous* credential pk_C. Concretely, this is achieved by obtaining the credential holder's *identity key* $\mathsf{pk}_{\mathcal{P}}$ which is linked to the party's identity $ID_{\mathcal{P}}$ via a public key infrastructure.

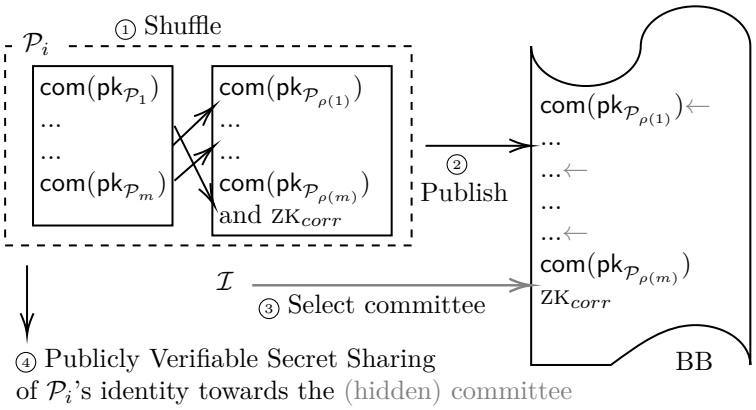


Figure 5.1: Mechanics of $\prod_{PC}$: ① Each user $\mathcal{P}_i$ locally generates commitments to hide each committee candidate's public key. Then, the party shuffles the set of commitments in a provable way (ZK$_{corr}$). ② The output of the shuffle is published on a public bulletin board (BB) by $\mathcal{P}_i$. ③ The issuer $\mathcal{I}$ selects the committee members for $\mathcal{P}_i$ from the shuffled list. ④ $\mathcal{P}_i$ secret shares its identity towards the selected committee members in a publicly verifiable way.

The Main Protocol The core idea in our main construction of PAPR anonymous credentials is to enable users to sample a random and anonymous committee in a verifiable way, using a verifiable shuffle. The protocol leverages a Public Key Infrastructure where keys for all m users are registered. Intuitively, to establish an anonymous committee, a user commits to all user public keys in the list, shuffles (*i.e.*, permutes and re-randomizes) the initial commitments and proves that it has done so correctly, posting the resulting commitments and proof to a Public Bulletin Board (BB). The issuer then selects the committee from the shuffled commitments by publishing $n < m$ random indices on the BB. This approach to committee selection is illustrated in Figure 5.1.

A credential request requires the user to publish secret shares of its identity encrypted under the public key of the selected committee along with zero knowledge proofs of share validity (*i.e.*, providing a publicly verifiable secret sharing of its identity). This creates a link between the credential and the encrypted shares of the identity, without revealing which identity was shared.

Since the issuer cannot learn the identity of the members of the revocation committee, it can only trigger privacy revocation for any issued credential by posting a public request on the BB. The committee members, monitoring the BB, reacts to such a request and proceed to reconstruct the user's identity by providing the decrypted shares to the issuer via a private channel.

We stress that both during committee establishment and secret sharing to the committee, all computation and communication is carried out by the user and the issuer only, without involving the committee members at all.

In this protocol, differently from the YOSO model, we allow the party who

requests a credential to learn the identities of the corresponding committee members. The rationale is that, as far as static security is concerned, an adversary playing as a malicious user can already link the identity of a corrupted committee member to an anonymous credential. Letting the identities of the elected committee members be known to the requesting party in this way creates no incentive of corruption, as it leaks no additional information. We stress that while the identities of committee members are learned, the selecting party still has no influence over what parties constitute the committee since they are selected provably at random.

Proactively Secure Versions Our main protocol is only secure against static adversaries. To withstand mobile adversaries, who can periodically uncorrupt parties and corrupt new parties, a heavier machinery is needed. It is crucial to notice that mobile adversaries in our setting can 1) corrupt a majority of the committee holding revocation data for a corrupted party's credential, which would allow an adversary to block privacy revocations, and 2) gradually corrupt a majority of the committee holding revocation data for an honest party (by moving to a new disjoint set of parties every epoch), which would allow it to stealthily learn the honest party's identity. Such mobile adversaries could be trivially addressed by computing the steps for issuing and revoking a credential via YOSO MPC, where each round of the computation is performed by a fresh randomly chosen fully anonymous committee, preventing the adversary from corrupting the committee currently holding the computation's secret state. However, YOSO MPC is notoriously expensive. Therefore, as a first step towards security against a mobile adversary, we instead show that we can use proactive secret sharing in the YOSO model, where committees are not known to *any* party, and the shared revocation data is periodically transferred to a new randomly chosen anonymous committee. While this technique solves the issue in a simple way, it requires the YOSO committees to hold an amount of data linear in the number of credentials issued.

An even more efficient alternative for proactive security is to employ YOSO threshold encryption and adding distributed key generation to our setup phase to obtain a system wide encryption public key. Issuance is then modified so that each party publishes an encryption of its identity under this common encryption key and proves in zero knowledge that they have done so in a way that creates a link between this encryption and the issued credential. Revocation can be done later by threshold-decrypting the ciphertext connected to that credential. The advantages of the latter approach are twofold, it both makes credential issuance simpler for parties (*i.e.*, they generate one ciphertext instead of encrypting multiple shares), and improves communication complexity for the YOSO committee members, since they only have to hold shares on a single secret key.

5.1.1 Cryptographic Primitives

Our construction employs a key-private encryption scheme (*i.e.*, an encryption which hides the recipient's public key) $\mathsf{Enc} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$,

a signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, a commitment scheme $\mathsf{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$, and Shamir Secret Sharing [162]. We further use two special types of digital signature schemes, structure preserving signatures [3] defined as $\mathsf{SPSig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ (similarly to Sig), and blind signatures [152] defined as $\mathsf{BSig} = (\mathsf{KeyGen}, \mathsf{User}, \mathsf{Sign}, \mathsf{Verify})$. Details on these schemes are presented in Section 2.

Structure preserving signatures are digital signatures where signatures σ and messages m belong to the same space. Blind signatures are a variant of signatures where the signer does not learn the message she signs. In known constructions the blind signature generation procedure is an interactive protocol between the signer and the party wishing to have a message signed.

We use a non-interactive zero-knowledge (NIZK) proof of shuffle correctness for commitments defined as the triple of algorithms $\mathsf{Shuf} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as per Definition 6. This NIZK allows for proving that a certain (public) vector of commitments was obtained by re-randomizing a given (public) vector of commitments and permuting the re-randomized commitments without revealing the randomness used for re-randomization nor the permutation. This NIZK can be efficiently realized from the proof of shuffle correctness for ciphertexts of [23]. In our setting, we view an ElGamal ciphertext as a commitment and use proofs of commitment shuffle correctness to convince a verifier that two distinct sets of commitments yield the same set of openings. The definitions of completeness, soundness and zero-knowledge for Shuf follow the same structure and aims as in [23] and presented in Section 2.

5.1.2 Ideal Functionalities

We make use of a set of ideal functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$. These functionalities are formally defined in Section 2. Briefly, the bulletin board functionality $\mathcal{F}_{BB}$, works so that any party can publish a message m to the board by sending (POST, sid, m) and read the contents of the board by sending (READ, sid). $\mathcal{F}_{PKI}$ is a functionality where each party can only send (POST, sid, m) once and can retrieve party $\mathcal{P}$'s message as $(\text{READ}, sid, \mathcal{P})$. The functionality for interactive zero knowledge, $\mathcal{F}_{ZK}$ is defined so that a prover $\mathcal{P}$ can send $(\text{ZK-PROVER}, sid, \mathcal{V}, x, w)$ to $\mathcal{F}_{ZK}$, which sends $(\text{ZK-PROOF}, sid, x)$ to the verifier $\mathcal{V}$ only if w is a witness for the statement x. Analogously, the functionality for non-interactive zero knowledge $\mathcal{F}_{NIZK}$ is defined by $(\text{PROVE}, sid, x, w)$, returning a proof π guaranteeing that w is a witness for the statement x, and $(\text{VERIFY}, sid, x, \pi)$, outputting 1 for a valid π for the statement x.

5.2 Defining PAPR for Anonymous Credentials

In this section we introduce the notion of a Publicly Auditable Privacy Revocation (PAPR) Anonymous Credential Scheme and describe an ideal functionality $\mathcal{F}_{PC}$ for it. Section 5.3 presents our protocol Π_{PC} that realizes $\mathcal{F}_{PC}$ based on

efficient and well-known building blocks. Section 5.3.1 proves Π_{PC} secure in the presence of a static, malicious adversary in the UC framework [60].

Defining PAPR Credentials We define the notion of PAPR credentials as the ideal functionality $\mathcal{F}_{PC}$ presented in Figure 5.2. This functionality provides standard anonymous credential interfaces supporting requesting credentials (CRED-REQ), issuing credentials (ISSUE-CRED), and showing credentials (SHOW-CRED). While any party may request a credential, only a special party called the *issuer* may approve such a request. As usual, requesting a credential and later showing it does not reveal any information about the credential owner's identity to the issuer nor to the party who is shown a credential. However, we do not aim at achieving unlinkability across multiple credential showings. In order to capture the novel PAPR property, the identity revocation interface (ANNOUNCE-REV) allows the issuer to request the identity of the owner of a given credential at any time, but this also immediately informs all other parties that privacy has been revoked for that credential.

Functionality $\mathcal{F}_{PC}$

$\mathcal{F}_{PC}$ is parameterized by a credential space $\mathcal{PK}$. The functionality interacts with a set of users $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$, a special party called the issuer $\mathcal{I} = \mathcal{P}_{m+1}$ and the ideal adversary $\mathcal{S}$. It keeps a list L_{cred} of credentials and a setup list L_{setup}, both initialized to $\emptyset$.

Setup: On input (SETUP, *sid*) from $\mathcal{P}_i$, add that party to the list L_{setup}.

Credential Request: On input (CRED-REQ, *sid*) from $\mathcal{P}_i$, if $L_{setup} \neq \mathcal{P} \cup \mathcal{I}$, then ignore the request. If $\mathcal{P}_i$ is honest, sample a random pk_{C_i} from $\mathcal{PK}$ and send (CRED-REQ, *sid*) to $\mathcal{S}$. Otherwise send (KEY?, *sid*) to $\mathcal{S}$ and await response (KEY, *sid*, pk_{C_i}). Finally write $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 0)$ to L_{cred} and send (CRED-REQ, *sid*, pk_{C_i}) to $\mathcal{I}$.

Credential Issuance: On input (ISSUE-CRED, *sid*, pk_{C_i}) from $\mathcal{I}$, if $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 0) \in L_{cred}$, update the entry to $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1)$ and send (CRED, *sid*, pk_{C_i}) to $\mathcal{P}_i$ and $\mathcal{S}$. Else write $(\perp, \mathsf{pk}_{C_i}, 1)$ to L_{cred}.

Credential Showing: On input (SHOW-CRED, *sid*, $\mathsf{pk}_{C_i}, \mathcal{P}_j$) from $\mathcal{P}_i$, if $(\cdot, \mathsf{pk}_{C_i}, 1) \notin L_{cred}$, ignore the request. Send (VALID-CRED, *sid*, pk_{C_i}) to $\mathcal{P}_j$ and (VALID-CRED, *sid*, $\mathsf{pk}_{C_i}, \mathcal{P}_j$) to $\mathcal{S}$.

Privacy Revocation: On input (ANNOUNCE-REV, *sid*, pk_{C_i}) by $\mathcal{I}$, send (ANNOUNCE-REV, *sid*, pk_{C_i}) to all $\mathcal{P}_j \in \mathcal{P}$ and $\mathcal{S}$. If $(\cdot, \mathsf{pk}_{C_i}, 1) \notin L_{cred}$, then ignore the request. If $(\perp, \mathsf{pk}_{C_i}, 1) \in L_{cred}$, then delete $(\perp, \mathsf{pk}_{C_i}, 1)$ from L_{cred}, and ignore the request. Else, $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1) \in L_{cred}$, then delete $(\mathcal{P}_i, \mathsf{pk}_{C_i}, 1)$ from L_{cred}, output (IDENTITY, *sid*, $\mathcal{P}_i$, pk_{C_i}) to $\mathcal{S}$ and send a delayed output (IDENTITY, *sid*, $\mathcal{P}_i$, pk_{C_i}) to $\mathcal{I}$.

Figure 5.2: Ideal functionality $\mathcal{F}_{PC}$ for PAPR Credentials.

5.3 Realizing PAPR for Anonymous Credentials

In Figures 5.3 and 5.5 we describe protocol $\prod_{PC}$ for anonymous credentials with PAPR. We consider *malicious* adversaries that may deviate from the protocol in any arbitrary way. Moreover, in this section we consider the *static* case, where the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. We assume that parties have access to synchronous communication channels, *i.e.*, all messages are delivered with a known maximum delay. To be concise, in the protocol description we let all reads from $\mathcal{F}_{BB}$ and $\mathcal{F}_{PKI}$ be implicit. It is also implicit that if a variable that is part of a procedure (*e.g.* a public key) is not yet available on $\mathcal{F}_{PKI}$ or $\mathcal{F}_{BB}$, the current procedure will terminate without output (*i.e* ignore the procedure call). Lastly, to avoid undefined behaviour while keeping the protocol description simple, whenever more than one valid message with equal values exist on $\mathcal{F}_{BB}$, only the chronologically first message shall be considered. We further assume that a user remains anonymous when posting to $\mathcal{F}_{BB}$ as is the case in the YOSO model.

Using Committees

We assume that revocation committees are formed by selecting uniformly at random the smallest number n of parties from set $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ such that every committee is guaranteed an *honest majority* with overwhelming probability given a certain corruption ratio. Selecting committees in this way has been explored extensively in [87], where concrete numerical examples of its size are provided. Indeed, a few examples are available in Section 5.5.

Since all parties are potential committee members, they are expected to monitor the bulletin board. Notice, however, that our protocol works with revocation committees selected from any set of parties (potentially disjoint from the set of parties who request credentials, as discussed in Section 5.5.2) as long as these committee have honest majority with overwhelming probability.

Protocol Overview

We now give a step-by-step overview of protocol $\prod_{PC}$.

Setup The setup phase consists of enrolling keys for the parties in the system. Note that, by registering its identity key $\mathsf{pk}_{\mathcal{P}_i}$ to the PKI, the user key and identity are linked. This link forms the basis for user identification during privacy revocation.

Committee Establishment Before a credential can be issued, a committee with which each party's identity will be shared must be established. Each party first executes the **Hide Committee Candidates** procedure. In step (1) the party hides the order of the committee candidates using a verifiably random

Protocol $\prod_{PC}$ (First Part)

Protocol $\prod_{PC}$ is executed by an issuer $\mathcal{I}$ and parties $\mathcal{P}_i \in \{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ interacting with functionalities $\mathcal{F}_{PKI}$, $\mathcal{F}_{BB}$, $\mathcal{F}_{NIZK}$ and $\mathcal{F}_{ZK}$. $\prod_{PC}$ is parameterized by a constant $n \in \mathbb{Z}$ such that sampling n parties out of $\{\mathcal{P}_1, \ldots, \mathcal{P}_m\}$ yields an honest majority except with negligible probabilty.

Setup: The issuer $\mathcal{I}$ and all parties $\mathcal{P}_i$ proceed as follows:

1. (Issuer Setup) On input (SETUP, sid), $\mathcal{I}$ generates a blind signature keypair $(\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{BSig.KeyGen}(1^\lambda)$ with $\lambda \in \mathbb{N}$ being a security parameter, an enrollment keypair $(\mathsf{pk}_E, \mathsf{sk}_E) \leftarrow \mathsf{SPSig.KeyGen}(\mathsf{pp})$, a revocation keypair $(\mathsf{pk}_R, \mathsf{sk}_R) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and an issuance keypair $(\mathsf{pk}_I, \mathsf{sk}_I) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and sends $(\text{REPORT}, sid, (\mathsf{pk}_B, \mathsf{pk}_E, \mathsf{pk}_R, \mathsf{pk}_I))$ to $\mathcal{F}_{PKI}$.
2. (User Setup) On input (SETUP, sid), $\mathcal{P}_i$ generates user identifying keys $(\mathsf{pk}_{\mathcal{P}_i}, \mathsf{sk}_{\mathcal{P}_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and sends $(\text{REPORT}, sid, \mathsf{pk}_{\mathcal{P}_i})$ to $\mathcal{F}_{PKI}$. Additionally $\mathcal{P}_i$ generates a single-use token keypair $(\mathsf{pk}_{T_i}, \mathsf{sk}_{T_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$ and interacts with $\mathcal{I}$ over a secure channel to obtain signatures $\sigma_E(\mathsf{pk}_{\mathcal{P}_i}) \leftarrow \mathsf{SPSig.Sign}(\mathsf{sk}_E, \mathsf{pk}_{\mathcal{P}_i})$. Finally $\mathcal{P}_i$ runs $\mathsf{BSig.User}(\mathsf{pk}_B, \mathsf{pk}_{T_i})$ with $\mathcal{I}$ running $\mathsf{BSig.Sign}(\mathsf{sk}_B)$ so as to compute the blind signature $\sigma_B(\mathsf{pk}_{T_i})$.
3. (Hide Committee Candidates) Let $\vec{\mathsf{pk}}_{\mathcal{P}}$ be the vector of all $\mathsf{pk}_{\mathcal{P}_j}$ and $\vec{f}$ the vector, s.t. $\vec{f}[j] = \mathsf{C.Commit}(\vec{\mathsf{pk}}_{\mathcal{P}}[j], 1)$, then $\mathcal{P}_i$ proceeds as follows:
   (a) Sample a random permutation ρ_i and verifiably shuffle $\vec{f}$ as $(\vec{f}'_i, \pi_{\rho_i}) = \mathsf{Shuf.Prove}(m, \rho_i, \vec{f})$. Sign the shuffle as $\sigma_{T_i}(\vec{f}'_i) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{T_i}, \vec{f}'_i)$,
   (b) Send $(\text{POST}, sid, (\texttt{hide}, \vec{f}'_i, \pi_{\rho_i}, \mathsf{pk}_{T_i}, \sigma_{T_i}(\vec{f}'_i), \sigma_B(\mathsf{pk}_{T_i}))$ to $\mathcal{F}_{BB}$.
4. (Sample Committee) When $(\text{POST}, sid, (\texttt{hide}, \vec{f}'_j, \pi_{\rho_j}, \mathsf{pk}_{T_j}, \sigma_{T_j}(\vec{f}'_j), \sigma_B(\mathsf{pk}_{T_j}))$ appers on $\mathcal{F}_{BB}$, $\mathcal{I}$ proceeds as follows:
   (a) Check that $1 = \mathsf{BSig.Verify}(\mathsf{pk}_B, \mathsf{pk}_{T_j}, \sigma_B(\mathsf{pk}_{T_j}))$, $1 = \mathsf{Sig.Verify}(\sigma_{T_j}(\vec{f}'_j))$ and $1 = \mathsf{Shuf.Verify}(m, \vec{f}, \vec{f}'_j, \pi_{\rho_j})$.
   (b) If true, let $\vec{b}_j$ where $\vec{b}_j[i] \xleftarrow{\$} \mathbb{Z}^*_{m+1}, |\vec{b}_j| = n$, indicate the indexes selected for the committee, sign $\vec{b}_j$ as $\sigma_I(\vec{b}_j) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_I, \vec{b}_j)$, send $(\text{POST}, sid, (\texttt{sample}, \vec{f}'_j, \vec{b}_j, \sigma_I(\vec{f}'_j || \vec{b}_j)))$ to $\mathcal{F}_{BB}$ and store $(\vec{f}'_j, \vec{b}_j)$ internally.

Credential Request: On input $(\text{CRED-REQ}, sid)$, if there is an entry $(\texttt{sample}, \vec{f}'_j, \vec{b}_j, \sigma_I(\vec{f}'_j || \vec{b}_j))$ where $1 = \mathsf{Sig.Verify}(\mathsf{pk}_I, \sigma_I(\vec{f}'_j || \vec{b}_j)$ on $\mathcal{F}_{BB}$, $\mathcal{P}_i$ proceeds as follows:

1. Define $\vec{a}_i[j] = \vec{\mathsf{pk}}_{\mathcal{P}}[\rho(j)]$, $\vec{h}_i[j] = \vec{a}_i[\vec{b}_i[j]]$ and $\vec{c}_i[j] = \vec{f}'_i[\vec{b}_i[j]]$ for $j = 1, \ldots, n$.
2. Generate identity shares via Shamir secret sharing, *i.e.*, sample a random polynomial $f()$ of degree $\lceil \frac{n}{2} \rceil$ where $f(0) = \mathsf{pk}_{\mathcal{P}_i}$ and set $\vec{s}_i[j] = f(j)$ for $j = 1, \ldots, n$. Encrypt the shares under the committee public keys obtaining $\vec{E}_i[j] = \mathsf{Enc.Encrypt}(s_i[j], \vec{h}_i[j])$ and construct committee member indicators $\vec{q}_i[j] \leftarrow \mathsf{Enc.Encrypt}(\vec{h}_i[j], \vec{h}_i[j])$ for $j = 1, \ldots, n$.
3. Prove correct escrow by sending $(\text{PROVE}, sid, x_i, w)$ to $\mathcal{F}_{NIZK}$ and getting $(\text{PROOF}, sid, \pi_{esc_i})$, where x_i and w are defined as in ZK_{esc} (Figure 5.4).
4. Generate user credential keys $(\mathsf{pk}_{C_i}, \mathsf{sk}_{C_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$.
5. Sends $(\text{POST}, sid, (\texttt{req}, \vec{E}_i, \vec{f}'_i, \vec{q}_i, \mathsf{pk}_{C_i}, x_i, \pi_{esc_i}))$ to $\mathcal{F}_{BB}$.

Figure 5.3: $\prod_{PC}$ - Setup, Committee Establishment and Credential Request.

shuffle, and is then (anonymously) bound to the shuffle by signing it with sk_T. In step (2), it publishes the shuffle, proof, and signature on the bulletin board.

The issuer then in step (1) of the *Sample Committee* procedure verifies that the requesting party has published a single signed and valid shuffle. If so, in step (2) it responds with a set of random indexes, indicating which of the shuffled values in $\vec{f}'$ shall constitute the committee.

Credential Issuance In the **Credential Request** procedure, a user in step (1) collects the public keys of the committee as indicated by $\mathcal{I}$ into $\vec{h}_i$, It also puts the corresponding commitments to the committee keys into $\vec{c}_i$. It then in step (2) produces a vector of encrypted shares $\vec{E}_i$ of its enrolled identity public key $\mathsf{pk}_{\mathcal{P}_i}$ for the committee in $\vec{h}_i$. To allow other users to known whether they are in the committee, a set of indicators, $\vec{q}_i$, is also produced. A party knows it is the j'th member of a committee if $\vec{q}_i[j]$ decrypts to its public key. Before generating credential keys in step (4) and posting the credential request in step (5), a party must first prove correct sharing in step (3). We provide a detailed description of the proven relation ZK_{esc} in the next subsection below.

When the issuer observes a credential request on the bulletin board it first executes step (1) of the **Credential Issuance** procedure to verify that a committee has been formed. Step (2) is executed to verify that sharing is done correctly by the requesting user. If all checks pass, step (3) is executed to sign the credential and publish it.

$$\text{ZK}_{esc}\{\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}), \vec{h}, \vec{s}, \vec{r} \mid \text{ZK}_{ID} \wedge \text{ZK}_{share}\}$$

$$\begin{array}{ll}
\text{① } \text{ZK}_{ID}\{\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}) \mid & \text{② } \text{ZK}_{share}\{\vec{h}, \vec{s}, \mathsf{pk}_{\mathcal{P}} \mid \\
\mathsf{Sig.VerifyKey}(\mathsf{sk}_{\mathcal{P}}, \mathsf{pk}_{\mathcal{P}}) \wedge & \text{②.1 } \mathsf{pk}_{\mathcal{P}} = \mathsf{SShare.Reconstruct}(\vec{s}) \wedge \\
\mathsf{SPSig.Verify}(\mathsf{pk}_E, \mathsf{pk}_{\mathcal{P}}, \sigma_E(\mathsf{pk}_{\mathcal{P}}))\} & \text{②.2 } \forall j \in \{1, \ldots, n\} : \\
 & \text{②.3 } \vec{E}[j] = \mathsf{Enc.Encrypt}(\vec{s}[j], \vec{h}[j]) \wedge \\
 & \text{②.4 } \mathsf{C.Open}(\vec{c}[j], \vec{h}[j], \vec{r}[j]) \wedge \\
 & \text{②.5 } \vec{q}[j] = \mathsf{Enc.Encrypt}(\vec{h}[j], \vec{h}[j]) \ \}
\end{array}$$

Figure 5.4: Elements of the ZK_{esc} statement. Intuitively, ZK_{ID} states that the proving user controls the enrolled identity key $\mathsf{pk}_{\mathcal{P}}$. ZK_{share} states that the identity key pk_U has been correctly shared to the committee members in $\vec{h}$.

Proving Correct Escrow The correctness of the identity escrow in a credential request is defined by the relation ZK_{esc}. Figure 5.4 defines ZK_{esc} on a high level, *i.e.*, by using procedure definitions. To simplify notation, we here define a procedure for knowledge of a private key, $\mathsf{Sig.VerifyKey}(\mathsf{sk}, \mathsf{pk}) \rightarrow v$, which indicates if sk, pk is a valid keypair with respect to $\mathsf{Sig.KeyGen}(.)$.

For illustrative purposes, we define ZK_{esc} as a conjunction, where $\text{ZK}_{esc} = \{\text{ZK}_{ID} \wedge \text{ZK}_{share}\}$. The first part, ① ZK_{ID}, states that the prover is the owner

of $\mathsf{pk}_{\mathcal{P}}$, *i.e* it knows secret key $\mathsf{sk}_{\mathcal{P}}$, and an issuer signature, $\sigma_E(\mathsf{pk}_{\mathcal{P}})$, on $\mathsf{pk}_{\mathcal{P}}$. The second part, ② ZK_{share} is a statement that (2.1) the shares are constructed correctly, *i.e.*, any set of k shares will reconstruct to the users public key $\mathsf{pk}_{\mathcal{P}}$. Further, (2.2) each of these shares, (2.3) is correctly encrypted, (2.4) for the correct committee member, (2.5) which is correctly indicated in $\vec{q}$.

Credential Showing The **Credential Showing** and **Verify Credential Showing** procedures are straightforward zero knowledge proofs of knowledge of the credential private key sk_{C_i} for the public key pk_{C_i} (and when verifying, also checking that the shown credential has been issued by $\mathcal{I}$ and that the credential is not revoked).

Privacy Revocation To learn the secret identity behind a credential public key pk_{C_j}, *i.e.*, to revoke the privacy, the issuer (and only the issuer) can execute the **Request Privacy Revocation** procedure. This procedure consists of publishing an announcement of the request for privacy revocation, signed with the privacy revocation key. Any (honest) user $\mathcal{P}_i$, observing such a request executes the **Privacy Revocation Response** procedure, where it first checks that a credential exists for this credential in step (1). If so, in step (2) all committee member indicators in $\vec{q}_j$ of that request are checked by decrypting them with the responding users identity secret key $\mathsf{sk}_{\mathcal{P}_i}$. If decryption results in the users identity public key $\mathsf{pk}_{\mathcal{P}_i}$ for the k'th indicator, $\mathcal{P}_i$ holds the k'th seat in the committee. If so, it (3) decrypts the k'th share, (4) proves correct decryption and committee membership, and (5) encrypts both the share and proof (since the proof reveals the share) for the issuer, and (6) sends the ciphertexts to the issuer. The issuer, when receiving such a share, executes the **Reconstruct Revoked Identity** procedure to decrypt and check the proof. When it has obtained a mojority of the shares, it reconstructs the revoked identity and obtains $\mathsf{pk}_{\mathcal{P}_j}$.

5.3.1 Security Analysis of $\prod_{PC}$

We now prove that $\prod_{PC}$ realizes $\mathcal{F}_{PC}$ in the presence of a static malicious adversary capable of corrupting up to $\frac{m}{2} - 1$ users.

Theorem 3. *Let* Sig *be a signature scheme,* BSig *be a blind signature scheme,* SPSig *be a structure preserving signature scheme,* SShare *be a* $(t, n)-$*threshold secret sharing scheme,* C *be a commitment scheme,* Enc *be a key-private* IND-CPA*-secure public-key encryption scheme and* Shuf *be a zero-knowledge proof of shuffle correctness as specified in Section 2. Protocol* $\prod_{PC}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the* ($\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$, $\mathcal{F}_{NIZK}$)*-hybrid model with security against a static active adversary* $\mathcal{A}$ *corrupting a minority of* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *such that a committee of size* $n \leq m$ *has honest majority with overwhelming probability.*

Proof. Let $\mathcal{A}$ be a static adversary allowed to corrupt up to $m/2 - 1$ parties before the start of the execution, which remain corrupt throughout the execution.

Protocol $\prod_{PC}$ (Second Part)

Credential Issuance: On input (ISSUE-CRED, sid, pk_i), if there is an entry $(\mathtt{req}, \vec{E}_j, \vec{f}'_j, \vec{q}_j, \mathsf{pk}_{C_j}, x_i, \pi_{esc_i})$ on $\mathcal{F}_{BB}$, $\mathcal{I}$ does the following:

1. If no internal entry $(\vec{f}'_j, \vec{b}_j)$ exists, ignore the next steps.
2. Send (VERIFY, sid, x_j, π_{esc_j}) to $\mathcal{F}_{NIZK}$, await the reply (VERIFICATION, sid, v). If $0 = v$, ignore the next steps.
3. Send (POST, $sid, \mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_I, \mathsf{pk}_{C_j})$) to $\mathcal{F}_{BB}$.

Credential showing:

1. (Credential showing) On input (SHOW-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$), $\mathcal{P}_i$ proves ownership of pk_{C_i} by sending (ZK-PROVER, $sid, \mathcal{P}_j, x, \mathsf{sk}_{C_i}$) to $\mathcal{F}_{ZK}$ where x is a statement for the relation $\text{ZK}_{cred}\{\mathsf{sk}_{C_i} \mid \mathsf{Sig.VerifyKey}(\mathsf{sk}_{C_i}, \mathsf{pk}_{C_i})\}$.
2. (Verify Credential Showing) Any party, upon receiving (ZK-PROOF, sid, x) for ownership of pk_{C_j} from $\mathcal{F}_{ZK}$ additionally verifies that $\mathcal{F}_{BB}$ contains an entry $(\mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}))$ s.t. $\mathsf{Sig.Verify}(\mathsf{pk}_I, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j})) = 1$, and contains no entry $(\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$ s.t. $\mathsf{Sig.Verify}(\mathsf{pk}_R, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j})) = 1$.

Privacy Revocation:

1. (Request Privacy Revocation) On input (ANNOUNCE-REV, sid, pk_{C_j}), $\mathcal{I}$ requests privacy revocation for pk_{C_j}, by generating $\sigma_R(\mathsf{pk}_{C_j}) \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_R, \mathsf{pk}_{C_j})$ and sending (POST, $sid, (\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$), to $\mathcal{F}_{BB}$.
2. (Privacy Revocation Response) A user, $\mathcal{P}_i$, observing an entry $(\mathtt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j}))$ on $\mathcal{F}_{BB}$ (with a valid signature):
   (a) If no entries $(\mathtt{req}, \vec{E}_j, \cdot, \vec{q}_j, \mathsf{pk}_{C_j}, \cdot)$ and $(\mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j}))$ exists on $\mathcal{F}_{BB}$, ignore the next steps.
   (b) If for no k, $\mathsf{pk}_{\mathcal{P}_i} = \mathsf{Enc.Decrypt}(\vec{q}_j[k], \mathsf{sk}_{\mathcal{P}_i})$, ignore the next steps.
   (c) Calculates $s_k = \mathsf{Enc.Decrypt}(\vec{E}_j[k], \mathsf{sk}_{\mathcal{P}_i})$.
   (d) Constructs a statement x_k for the relation $\text{ZK}_{rev_k}\{\mathsf{sk}_{\mathcal{P}_i} \mid s_k = \mathsf{Enc.Decrypt}(\vec{E}_j[k], \mathsf{sk}_{\mathcal{P}_i}) \wedge \mathsf{pk}_{\mathcal{P}_i} = \mathsf{Enc.Decrypt}(\vec{q}_j[k], \mathsf{sk}_{\mathcal{P}_i}) \wedge \mathsf{Sign.VerifyKey}(\mathsf{sk}_{\mathcal{P}_i}, \mathsf{pk}_{\mathcal{P}_i}\})$ and sends the message (PROVE, $sid, x_k, \mathsf{sk}_{\mathcal{P}_i}$) to $\mathcal{F}_{NIZK}$ and await response (PROOF, sid, π_{rev_k}).
   (e) Encrypt π_{rev_k} and s_k for the issuer as $\tilde{s}_k \leftarrow \mathsf{Enc.Encrypt}(s_k, \mathsf{pk}_R)$, $\tilde{\pi}_{rev_k} \leftarrow \mathsf{Enc.Encrypt}(\pi_{rev_k}, \mathsf{pk}_R)$.
   (f) Sends a message (REV-SHARE, $sid, \mathsf{pk}_{C_j}, \tilde{s}_k, x_k \tilde{\pi}_{rev_k}$), to $\mathcal{I}$.
3. (Reconstruct Revoked Identity) Upon receiving a message (REV-SHARE, $sid, \mathsf{pk}_{C_j}, \tilde{s}_k, x_k \tilde{\pi}_{rev_k}$), $\mathcal{I}$ uses sk_I to decrypt $\tilde{s}_k$ and $\tilde{\pi}_{rev_k}$, to obtain s_k and π_{rev_k}, and verifies π_{rev_k} by sending (VERIFY, sid, x_k, π_{rev_k}) to $\mathcal{F}_{NIZK}$. On reply (VERIFICATION, $sid, 1$), $\mathcal{I}$ adds s_k to its internal set $S_{\mathsf{pk}_{C_j}-shares}$. If now $|S_{\mathsf{pk}_{C_j}-shares}| = \lceil \frac{n}{2} \rceil + 1$, calculate the revoked identity key by Lagrange interpolating the polynomial f' defined by the shares in $S_{\mathsf{pk}C_j-shares}$ and then calculate the identifying public key of the revoked user as $f'(0) = \mathsf{pk}_{\mathcal{P}_j}$.

Figure 5.5: $\prod_{PC}$ - Credential Issuance, Credential Showing and Privacy Revocation.

We prove Theorem 3 by showing that for each $\mathcal{A}$, there exists a simulator $\mathcal{S}_{PC}$ so that any environment $\mathcal{Z}$ has a negligible advantage in determining whether it is interacting with $\mathcal{A}$ and $\prod_{PC}$ or $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$. $\mathcal{S}_{PC}$ is described in Figures 5.6 and 5.7.

Indistinguishably of Setup In this step, the vectors $\vec{f}$ ($\vec{f}[j] = \mathsf{C.Commit}(\vec{\mathsf{pk}}_{\mathcal{P}}[j], 1)$) and $\vec{f}'_i$ ($(\vec{f}'_i, \pi_{\rho_i}) = \mathsf{Shuf.Prove}(m, \rho_i, \vec{f})$) are indistinguishable from those computed in a real execution due to the hiding property of commitments. Similarly, π_{ρ_i} is indistinguishable due to the zero knowledge property of zero knowledge proofs. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably of Credential Requests In this step, the simulated proof π_{esc_i} is indistiguishable from the one computed in a real execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{NIZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably Credential Issuance In this step it is simulated the creation of a credential without having any information about the identity of the honest party who requests the credential in the real world execution. However, π_{esc_i} is indistinguishable from the one computed in the real world execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{NIZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably of Credential Showings In this step it is simulated the showing of a credential without having any information about the identity of the honest party who shows it in the real world execution. However, (ZK-PROOF, sid, x) is indistinguishable from the one computed in the real world execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{ZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Indistinguishably of Privacy Revocation In this step, the simulated π_{rev_k}, computed for the adjusted shares s'_k, is indistinguishable from the one computed in the real world execution since $\mathcal{S}_{PC}$ perfectly emulates $\mathcal{F}_{NIZK}$. Thus, $\mathcal{Z}$ cannot distinguish this step of the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

Notice that throughout the simulation $\mathcal{S}_{PC}$ interacts with $\mathcal{A}$ exactly as an honest party would in $\prod_{PC}$, except when simulating credential issuance and showing for honest parties. In these cases, $\mathcal{S}_{PC}$ simulates the creation of a credential and its showing without having any information about the identity of the honest party who requests/shows the credential. However, this is indistinguishable from the real world execution since these proofs are done via $\mathcal{F}_{NIZK}$

and $\mathcal{F}_{ZK}$, which produces messages distributed exactly as in a real world execution. Moreover, by extracting witnesses from proofs done by $\mathcal{A}$ via $\mathcal{F}_{NIZK}$ and $\mathcal{F}_{ZK}$, $\mathcal{S}_{PC}$ activates $\mathcal{F}_{PC}$ with inputs that match $\mathcal{A}$'s behavior. Hence, $\mathcal{Z}$ cannot distinguish the ideal world execution with $\mathcal{S}_{PC}$ and $\mathcal{F}_{PC}$ from the real world execution of $\prod_{PC}$ with $\mathcal{A}$.

□

5.4 From Static to Proactive Security

Protocol $\prod_{PC}$ as described in the previous sections realizes a PAPR credential scheme using efficient building blocks, in the static security setting. In this section, we sketch how to construct proactively secure PAPR Credentials, at the price of using less efficient building blocks.

Maintaining the revocation committee secret in the presence of a mobile adversary naturally puts us in the YOSO setting: the identities of committee members must remain anonymous, so before they act in a revocation process (or before) the adversary moves, they must re-share the revocation information they hold towards a new anonymous committee. While it would be straightforward to design a protocol realizing $\mathcal{F}_{PC}$ by use of YOSO MPC, it would be terribly inefficient, since it would require computing our credential issuance procedure as part of a very complex YOSO MPC computation where a fresh anonymous committee performs each round. Instead, we propose two alternative and more efficient constructions. The first demonstrates how to wrap our protocol $\prod_{PC}$ with a YOSO resharing procedure to obtain proactive security. The second improves efficiency further by using YOSO Threshold Encryption directly.

5.4.1 Modeling Proactive Security

We model proactive security, similarly to [130], by each party in the system having an *epoch* tape which maintains an integer `epoch` initialized to 0 at the start of the execution. The execution proceeds in phases which alternate between an *operational* phase and a *refreshing* phase, starting with the operational phase. In contrast to [130], we force every party to have the same value as epoch counter.

Epochs The refreshing stage is started by the adversary sending `refresh` to *all* parties. Refresh of individual parties is not allowed. Upon receiving the `refresh` command, a party increases `epoch` by 1 and executes its instructions for refreshment. Once each party has completed its refreshment instructions and handed over execution to $\mathcal{Z}$, a new operational phase begins.

Corruptions A mobile adversary $\mathcal{A}$ can corrupt or uncorrupt any party $\mathcal{P}_i$ *after* a refreshing phase ends (*i.e.*, after the last party has handed over execution to $\mathcal{Z}$) but *before* the next operational phase starts (*i.e.*, before the first activation

Simulator $\mathcal{S}_{PC}$ (First Part)

$\mathcal{S}_{PC}$ interacts a copy of the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. $\mathcal{S}_{PC}$ forwards all messages between $\mathcal{A}$ and $\mathcal{Z}$. $\mathcal{S}_{PC}$ acts as $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ towards $\mathcal{A}$, by following their respective descriptions (unless otherwise explicitly stated).

Setup:

1. (Issuer Setup) Run the $\prod_{PC}$ procedure for **Issuer Setup** if the issuer is honest and send (SETUP, sid) to $\mathcal{F}_{PC}$.
2. (User Setup) Run the $\prod_{PC}$ procedure for **User Setup** for all honest users, then send (SETUP, sid) to $\mathcal{F}_{PC}$ for all honest users.

   For each (POST, $sid, \cdot$) sent to $\mathcal{F}_{PKI}$ by a corrupt party, send (SETUP, sid) to $\mathcal{F}_{PC}$.
3. (Hide Committee Candidates) Run the **Hide Committee Candidates** procedure for each honest user.
4. (Sample Committee) If $\mathcal{I}$ is honest, for each (POST, sid, ($\mathtt{hide}, \vec{f}'_j, \pi_{\rho_j}, \mathsf{pk}_{T_j}, \sigma_{T_j}(\vec{f}'_j), \sigma_B(\mathsf{pk}_{T_j})$) sent to $\mathcal{F}_{BB}$, (either by a corrupt $\mathcal{P}_j$ or when simulating an honest $\mathcal{P}_j$ in the previous paragraph), the simulator executes step (1) of the **Sample Committee** procedure. If the checks verify, also execute step (2).

Credential Request: If the simulator receives a message (CRED-REQ, sid) from $\mathcal{F}_{PC}$, an honest user has requested a credential. To simulate this, $\mathcal{S}_{PC}$ executes the **Credential Request** procedure in $\prod_{PC}$, but does so using arbitrary values for $\mathsf{sk}_{\mathcal{P}_i}, \mathsf{pk}_{\mathcal{P}_i}$ and $\sigma_E(\mathsf{pk}_{\mathcal{P}_i})$ and a simulated proof π_{esc_i} for the arbitrary values. If later $\mathcal{S}_{PC}$ receives a message for $\mathcal{F}_{NIZK}$, (VERIFY, sid, x_i, π_{esc_i}) it responds with (VERIFICATION, sid, 1) instead of following the $\mathcal{F}_{NIZK}$ description. If the simulator receives ($\mathtt{req}, \vec{E}_i, \vec{f}'_i, \vec{q}_i, \mathsf{pk}_{C_i}, x_i, \pi_{esc_i}$) from a corrupt user $\mathcal{P}_i$, intended for $\mathcal{F}_{BB}$, the simulator executes checks in step (1) and (2) of the **Credential Issuance** procedure. If the checks clear, the request is valid and the simulator then sends (CRED-REQ, sid) to $\mathcal{F}_{PC}$, awaits the message (KEY?, sid) and responds by sending (KEY, sid, pk_{C_i}) to $\mathcal{F}_{PC}$.

Credential Issuance: If $\mathcal{I}$ is honest, on message (CRED, sid, pk_{C_j}) from $\mathcal{F}_{PC}$ simulate by executing step (3) of the **Credential Issuance** procedure. If instead $\mathcal{I}$ is corrupt, for each message (POST, sid, $\mathtt{issue}, \mathsf{pk}_{C_j}, \sigma_I(\mathsf{pk}_{C_j})$) sent to $\mathcal{F}_{BB}$, $\mathcal{S}_{PC}$ sends (ISSUE-CRED, sid, pk_{C_j}) to $\mathcal{F}_{PC}$ if $\sigma_I(\mathsf{pk}_{C_j})$ is a valid signature on pk_{C_j} by $\mathcal{I}$.

Credential showing:

1. (Credential showing) Whenever the simulator receives (VALID-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$) from $\mathcal{F}_{PC}$, $\mathcal{S}_{PC}$ simulates a successful show of credential pk_{C_i} to party $\mathcal{P}_j$ by simulating a successful proof of relation ZK_{cred} via $\mathcal{F}_{ZK}$ with $\mathcal{P}_j$ acting as verifier, resulting in $\mathcal{P}_j$ receiving (ZK-PROOF, sid, x_i) from $\mathcal{F}_{ZK}$.
2. (Verify Credential Showing) Whenever the simulator receives a message (ZK-PROVER, $sid, \mathcal{P}_j, x_i, \mathsf{sk}_{C_i}$), where x_i is a statement for $\text{ZK}_{cred}\{\mathsf{sk}_{C_i} \mid \mathsf{Sig.VerifyKey}(\mathsf{sk}_{C_i}, \mathsf{pk}_{C_i})\}$, from a corrupt party $\mathcal{P}_i$, intended for $\mathcal{F}_{ZK}$, it verifies the proof by following the description of $\mathcal{F}_{ZK}$. If the verification clears, send (SHOW-CRED, $sid, \mathsf{pk}_{C_i}, \mathcal{P}_j$) to $\mathcal{F}_{PC}$.

Figure 5.6: Simulator $\mathcal{S}_{PC}$ for protocol $\prod_{PC}$.

Simulator $\mathcal{S}_{PC}$ (Second Part)

Privacy Revocation:

1. (Request Privacy Revocation) If the issuer is honest, and the simulator receives a message (ANNOUNCE-REV, sid, pk_{C_j}) from $\mathcal{F}_{PC}$, it executes the **Request Privacy Revocation** procedure in $\prod_{PC}$ to simulate the issuer. If the issuer is corrupt and $\mathcal{S}_{PC}$ receives (POST, $sid, \texttt{rev}, \mathsf{pk}_{C_j}, \sigma_R(\mathsf{pk}_{C_j})$), intended for $\mathcal{F}_{BB}$ where $\sigma_R(\mathsf{pk}_{C_j})$ is a valid revocation signature on pk_{C_j}, the simulator sends (ANNOUNCE-REV, sid, pk_{C_j}) to $\mathcal{F}_{PC}$.
2. (Reconstruct Revoked Identity) When $\mathcal{S}_{PC}$ receives a message with the identity of a user (IDENTITY, $sid, \mathcal{P}_i, \mathsf{pk}_{C_i}$), it must simulate responses from the honest committee members. If $\mathcal{I}$ is honest or $\mathcal{P}_i$ is corrupt, $\mathcal{S}_{PC}$ thus executes the $\prod_{PC}$ procedure for **Privacy Revocation Response** for each honest $\mathcal{P}_j$.

   If $\mathcal{I}$ is corrupt and $\mathcal{P}_i$ is honest the simulator needs to "adjust" the shares which the honest committee members respond with, so that the shares reconstruct to $\mathsf{pk}_{\mathcal{P}_i}$ rather that the arbitrary value used during simulation of the credential request. Therefore the simulator first constructs a polynomial f of degree $\lceil \frac{n}{2} \rceil - 1$ where $f(\mathsf{pk}_{\mathcal{P}_i}) = 0$ and $f(k) = \vec{s}_i[k]$ for each k where $\rho_i(k) \in \vec{b}_i$ (*i.e.*, for the corrupt users in the simulated committee, don't change the shares).

   Then for each k where $\rho_i(k) \notin \vec{b}_i$ (honest users) let $s'_k = f(k)$ and construct a statement x_k for relation ZK_{rev_k} with s'_k. If later the simulator receives a message for $\mathcal{F}_{NIZK}$, (VERIFY, sid, x_k, π_{rev_k}) it responds with (VERIFICATION, $sid, 1$) instead of following the $\mathcal{F}_{NIZK}$ description. Honest committee members are then simulated by executing step (5) and (6) of the **Privacy Revocation Response** procedure using s'_k and π_{rev_k} as constructed by the simulator.

Finally, when $\mathcal{A}$ stops, output whatever $\mathcal{A}$ outputs to $\mathcal{Z}$.

Figure 5.7: Simulator $\mathcal{S}_{PC}$ for protocol $\prod_{PC}$.

of a party in the operational phase). After $\mathcal{A}$ moves, every party $\mathcal{P}_i$ remains corrupted (or honest) throughout that entire operational phase. At no time can $\mathcal{A}$ corrupt more than $\lceil \frac{m}{2} \rceil - 1$ parties.

5.4.2 Proactive Security Through YOSO Resharing

Let us now describe how to modify $\prod_{PC}$ to obtain proactive security by adding a re-sharing procedure in the YOSO model. Resharing is a standard procedure in proactive secret sharing that allows a set of parties to transfer a shared secret for which they hold shares to a second set of parties who obtain fresh shares independent from the original ones. On a high level, YOSO resharing allows for a current committee to reshare a secret towards a future anonymous committee while only speaking once. Such a YOSO resharing procedure can be added to our PAPR protocol without modifying existing procedures. That is, we use $\prod_{PC}$ as it is, but add a YOSO reshare procedure for maintaining the escrowed user identities over different epochs. Before every new epoch starts, current revocation committees reshare the identity information they hold towards a single anonymous committee that holds this information in the next epoch. We

refer to this protocol as $\prod_{PC-P}$. The approach is illustrated in Figure 5.8.

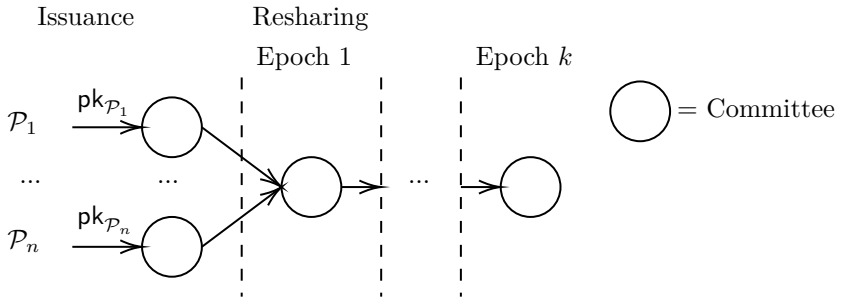


Figure 5.8: Functioning of $\prod_{PC-P}$ with YOSO resharing: as in the issuance procedure of $\prod_{PC}$, initially each user $\mathcal{P}_i$ secret shares its identity $\mathsf{pk}_{\mathcal{P}_i}$ towards a different designated hidden committee. Subsequently, the committees reshare the identities towards a new single anonymous committee and a resharing towards a new single anonymous committee is executed before the start of each upcoming epoch.

A YOSO resharing scheme can be abstractly described as having a **committee establishment** part, where all parties jointly elect the new committee without learning it, and a **resharing** part, where the current committee provably reshares the committee secret to the new committee without learning or revealing the new committee members. Multiple choices are available for implementing YOSO resharing, *e.g.* Evolving-Committee Proactive Secret Sharing [31], Random-Index Private Information Retrieval [104] plus standard resharing techniques, or YOLO YOSO Anonymous Committee PVSS Resharing [65]. We refrain from picking a particular scheme, and instead use the **committee establishment** and **resharing** procedures abstractly, as described below:

Committee Establishment During committee establishment, a single committee for the next epoch of size n is elected from all m committee candidates, without revealing the committee. This procedure will output a set of anonymous public keys which constitute the committee keys.

Resharing During resharing, each member of the current epoch committee re-shares the secret using the anonymous public keys of the next epoch's committee. This procedure will thus output a set of anonymously encrypted shares of the secret. Before these encrypted shares are published, the old shares must be made inaccessible, *e.g.* by deleting them.

Figure 5.9 describes how to add a refresh procedure based on YOSO-Resharing to $\prod_{PC}$ in order to realize $\mathcal{F}_{PC}$ proactive security against a mobile adversary $\mathcal{A}$. Protocol $\prod_{PC-P}$ is obtained by executing $\prod_{PC}$ with the modifications described in Figure 5.9 in order to securely refresh shares of revocation information across epoch changes. We here indicate instances of functionalities specific to

Hold Revocation Responses: Postpone revocation requests until *refresh* phase.
Reshare: $\mathcal{P}_i$ on command `refresh` from $\mathcal{Z}$ does:

(a) Generate new keys $(\mathsf{pk}'_{\mathcal{P}_i}, \mathsf{sk}'_{\mathcal{P}_i}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$, replace $\mathcal{F}_{PKI}^{\texttt{epoch}}$ with $\mathcal{F}_{PKI}^{\texttt{epoch + 1}}$ and send $(\text{POST}, sid, \mathsf{pk}'_{\mathcal{P}_i})$ to $\mathcal{F}_{PKI}^{\texttt{epoch + 1}}$.

(b) Execute the YOSO **Committee Establishment** procedure, obtaining the anonymous committee public keys for the `epoch + 1` committee.

(c) For each postponed revocation request for credentials issued in the current epoch, execute steps (1) to (5) of **Privacy Revocation Response** in $\prod_{PC}$, *i.e.*, stopping before sending shares to $\mathcal{I}$.

(d) If $\mathcal{P}_i$ is part of the YOSO committee for the current epoch, handle any revocation requests for credentials issued during previous epochs by executing steps (3) to (5) of the **Privacy Revocation Response** procedure in $\prod_{PC}$.

(e) Erase $\mathsf{sk}_{\mathcal{P}_i}$. Set $\mathsf{sk}_{\mathcal{P}_i} = \mathsf{sk}'_{\mathcal{P}_i}$, $\mathsf{pk}_{\mathcal{P}_i} = \mathsf{pk}'_{\mathcal{P}_i}$ and $\texttt{epoch} = \texttt{epoch} + 1$.

(f) For all credentials that have not been revoked, execute YOSO resharing of escrowed identities towards the `epoch + 1` committee. For all revocation requests handled in steps (c) or (d), post the results by executing step (6) of the **Privacy Revocation Response** procedure in $\prod_{PC}$.

Wrap: Any other input is forwarded to $\prod_{PC}$.

Figure 5.9: Sketch of proactive security wrapper protocol $\prod_{PC-P}$.

an epoch be indicated in the superscript, so that $\mathcal{F}_{PKI}^1$ is the shared instance during the first epoch and $\mathcal{F}_{PKI}^2$ the shared instance during the second.

Assuming an ideal functionality $\mathcal{F}_{YPSS}$ capturing YOSO proactive secret sharing with the properties outlined above, the security of $\prod_{PC-P}$ is captured as follows. Notice that such a $\mathcal{F}_{YPSS}$ can be obtained via the techniques of [103, 104, 65] plus UC-secure NIZKs modelled $\mathcal{F}_{NIZK}$.

Theorem 4. *(Informal) Let* Sig *be a signature scheme,* BSig *be a blind signature scheme,* SPSig *be a structure preserving signature scheme,* SShare *be a* $(t, n)-$*threshold secret sharing scheme,* C *be a commitment scheme,* Enc *be a key-private* IND-CPA*-secure public-key encryption scheme and* Shuf *be a zero-knowledge proof of shuffle correctness as specified in Section 2. Protocol* $\prod_{PC-P}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the* $(\mathcal{F}_{BB}, \mathcal{F}_{PKI}, \mathcal{F}_{ZK}, \mathcal{F}_{NIZK}, \mathcal{F}_{YPSS})$*-hybrid model, with proactive security against a mobile active adversary* $\mathcal{A}$ *corrupting a minority of parties in* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *so that any committee of size* $n \leq m$ *has honest majority, with overwhelming probability.*

5.4.3 Proactive Security Through YOSO Threshold Encryption

While the protocol in Figure 5.9 shows how to wrap $\prod_{PC}$ with a YOSO-resharing step to obtain proactive security, it is possible to realize a proactively secure PAPR credential scheme in a more efficient way using YOSO Threshold Encryption [92]. We can realize a PAPR Credential scheme assuming we

have such a YOSO Threshold encryption system, with procedures for setting up YOSO committees (*Committee Selection*), generating a committee keypair so that all system parties hold the public key and each committee member holds a share of the corresponding secret key (*Distributed Key Generation*), resharing the secret key (*Reshare*), decryption of a ciphertext to a share of the plaintext (*Threshold Decryption*) and reconstruction of the plaintext given a sufficient amount of shares of the plaintext (*Reconstruct*). We sketch our protocol $\prod_{PC-PT}$ below:

Setup Each party $\mathcal{P}_i$ generates an identity keypair and registers the public key on a PKI. The issuer $\mathcal{I}$ generates issuance and revocation keypairs, registers the public keys on a PKI and publishes signatures of each user's public key under the issuance key. All $\mathcal{P}_i$ execute the *Committee Selection* and the anonymous committee executes the *Distributed Key Generation* procedure obtaining a threshold public key pk_{THE} and shares of the corresponding secret key.

Credential Issuance To request a credential, a user generates a new credential keypair, encrypts its identity public key under pk_{THE}. It then sends this ciphertext and the public key of the new credential keypair to the issuer over an anonymous channel and proves in zero knowledge that it knows the private key and issuer signature on the encrypted public key. If the issuer accepts the proof, it returns a signature on the credential public key.

Revocation Request The issuer requests privacy revocation for a credential by signing the credential public key with its revocation key and posting the signature on a bulletin board.

Reshare and Revocation Response On command `refresh` from $\mathcal{Z}$, all current epoch honest committee members constructs revocation responses for privacy revocation requests correctly posted on the system bulletin board by executing the *Threshold Decryption* procedure to obtain shares of the revoked users identity public key. They then execute the committee *Reshare* procedure before giving the shares to the issuer. When the issuer obtain these shares, it learns the identity key of the revoked user by executing the *Reconstruct* procedure.

Assuming an ideal functionality $\mathcal{F}_{YTHE}$ capturing YOSO threshold encryption with the properties outlined above, the security of $\prod_{PC-PT}$ is captured as follows. Notice that such a $\mathcal{F}_{YTHE}$ can be obtained via the techniques o [92] by employing UC-secure NIZKs as modelled in $\mathcal{F}_{NIZK}$ and UC-secure proactive resharing as modelled in $\mathcal{F}_{YPSS}$ (discussed above).

Theorem 5. *(Informal) Let* Sig *be a signature scheme,* BSig *be a blind signature scheme and* Enc *be a key-private* IND-CPA*-secure public-key encryption scheme. Protocol* $\prod_{PC-PT}$ *UC-realizes* $\mathcal{F}_{PC}$ *in the* ($\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$, $\mathcal{F}_{NIZK}$, $\mathcal{F}_{YTHE}$)*-hybrid model with proactive security against a mobile active adversary* $\mathcal{A}$ *corrupting a minority of* $\mathcal{P}_1, \ldots, \mathcal{P}_m$ *such that a committee of size* $n \leq m$ *has honest majority with overwhelming probability.*

The advantage of this approach in relation to the simple extension $\prod_{PC-P}$ using YOSO resharing is that using YOSO threshold encryption in this way gives us amortized communication complexity essentially independently from the number of credentials issued. Notice that in $\prod_{PC-P}$ the YOSO committees are required to hold shares of the identity public keys connected to every credential that has been issued (and not revoked). On the other hand, in this improved construction, the YOSO committees only need to hold shares of the secret key for the threshold encryption scheme. Moreover, credential issuance also becomes cheaper, since a party who requests a credential no longer needs to secret share its identity public key towards a committee. In the new credential issuance procedure, a party only needs to publish an encryption of its identity public key under the threshold encryption public key, which also makes the zero-knowledge proof it generates in this phase cheaper (*i.e.*, proving that a single ciphertext contains a certain message, instead of proving that a set of encrypted secret shares reconstruct that message).

5.5 Practical Considerations

We now discuss the properties of PAPR for anonymous credential schemes from a practical perspective.

5.5.1 Optimizing the Size of the Committee

Given a set of parties $\mathcal{P}$ of size m and a certain corruption ratio t, we are interested in sampling uniformly at random the minimum number of parties n from $\mathcal{P}$ such that an honest majority committee is guaranteed with overwhelming probability $1 - 2^{-\lambda}$, where λ is a security parameter. This situation is extensively described in [87], but to aid intuition we here provide a few numerical examples when $\lambda = 60$. If $m = 10,000$ and $t = 30\%$, then $n = 462$. If $m = 2,000$ and $t = 30\%$, then $n = 382$. If $m = 10,000$ and $t = 20\%$, then $n = 178$. If $m = 2,000$ and $t = 20\%$, then $n = 164$.

5.5.2 Flexibility in the Protocol Design

Throughout this work we made some simplifying assumptions to ease the explanation. Below, we discuss ways to generalize our protocol in the cases where the assumptions are not actual limitations of the protocol design.

Multiple Authorities

The $\mathcal{F}_{PC}$ functionality and its concrete realization, $\prod_{PC}$, are defined for a single issuer $\mathcal{I}$. This is done to keep the protocol simple and easy to read. Extending the scheme to multiple authorities can be done straightforwardly in two ways. One way is to exploit the fact that the scheme is proven to be universally composable, so we can run multiple parallel instances without compromising security. This approach requires no changes to the functionality or the protocol

description. A second way is to define $\mathcal{F}_{PC}$ for multiple issuing parties. This can be done by imposing that credential requests shall specify which $\mathcal{I}$ can issue and revoke this credential, and by letting credential showings be valid for any issuing $\mathcal{I}$. This change can be trivially reflected in our $\prod_{PC}$ construction.

Separating the Issuance and Revocation Roles

Analogously to the previous paragraph, we have kept the protocol description simple by appointing a single party $\mathcal{I}$ for both issuance and revocation roles. Modifying $\mathcal{F}_{PC}$ and $\prod_{PC}$ by introducing a revoking party $\mathcal{R}$, and appointing the revocation role to $\mathcal{R}$, rather than $\mathcal{I}$, is straightforward: in $\mathcal{F}_{PC}$ allow $\mathcal{R}$ (instead of $\mathcal{I}$) to send (ANNOUNCE-REV, $sid, \cdot$). In $\prod_{PC}$ move the generation and PKI-registration of the revocation keypair $(\mathsf{pk}_R, \mathsf{sk}_R)$ into a separate **Revoker Setup** procedure, and in the **Privacy Revocation Response** procedure, send the shares to $\mathcal{R}$ rather than to $\mathcal{I}$. This separation of roles can be combined with the above modification for multiple authorities to freely select a desired set of issuers and revokers.

Establishing Eligible Committee Candidates

In PAPR, the set of committee candidates is the root of trust for the guaranteed privacy revocation and public announcement. In practice, our system can easily be adapted to have the list of eligible committee candidates be publicly chosen and endorsed, *e.g.*, through an election or by the issuer. In particular, the set of committee candidates does not have to coincide with the whole set of users.

Separating Users and Committee Candidates

In Section 5.3 we described $\prod_{PC}$ assuming the set of users and the set of committee candidates to be the same. This was a simplifying assumption, but it is not a limitation of our design. Indeed, $\prod_{PC}$ can be modified to accommodate a set of committee candidates that is independent from the set of users. For instance, split $\mathcal{P}$ into a subset $\mathcal{C} = \{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_c}\}$ of potential committee members and a subset of standard users $\mathcal{U} = \mathcal{P} \setminus \mathcal{C}$, and run the instructions **Hide Committee Candidates** and **Sample Committee** (from Figure 5.3), letting the index run among the public keys in $\mathcal{C}$. Committee candidates may be expected to be online all the time. This behavior can be incentivized through a reward system or law constraints. On the other hand, users are allowed to be offline whenever they wish.

Managing a dynamic user set

$\prod_{PC}$ crucially relies on $\mathcal{F}_{PKI}$ to contain a fixed list of all parties before credentials are issued. In practice the set of active users might however change over time, with users joining or leaving the system. However, this reliance is not as strong as it appears on first glance.

By running parallel instances of $\prod_{PC}$ with multiple authorities, as described above, each new instance will have a separate $\mathcal{F}_{PKI}$. Thus users joining an already existing system can be enrolled to a new instance of the protocol.

On the other hand, if enough committee candidates leave the system, *e.g.*, due to loss of their keys, the possibility of privacy revocation can be affected. While a party leaving the system would technically fall under corrupt behaviour, this is not a problem in $\prod_{PC-P}$ and $\prod_{PC-PT}$. This is since these protocols re-share committee secrets and explicitly use a new instance of $\mathcal{F}_{PKI}$ for each epoch. Thus, inactive users will not enroll with the new $\mathcal{F}_{PKI}$ and will as a consequence not be considered committee candidates anymore. In the case of $\prod_{PC}$ however, this mechanism is not present, and one must therfore account for the probability of parties leaving the system when selecting the size of n.

5.5.3 Overhead From a User Perspective

Despite the many parts of the protocol, from a user perspective, the protocol is a very low cost endeavor. $\prod_{PC}$ is designed with user overhead in mind, reducing complexity for the user and keeping as much of the resulting complexity in the credential issuance phase. A user only needs to store a bare minimum of their own identity key and their own credentials. Credential issuance is somewhat computationally intense for the user, but this only happens once – per credential issuance. During normal (application) operation, there is *zero* computational overhead for the user. Finally, a user will experience some additional computational overhead *when and only if* they are involved as a committee member in an actual privacy revocation request (or in a YOSO-resharing for $\prod_{PC-P}$). So in summary, computational efforts for users are only necessary in the beginning and sometimes (or rarely) at the end of an epoch, but never during normal operation.

5.5.4 Practical Attacks

Denial of Service

An adversary with the capability to mount large scale Denial of Service (DoS) attacks, *i.e.*, targeting all potential committee members, can of course delay privacy revocation while the attack is maintained. However, it cannot prevent revocation indefinitely. Once the DoS attack is mitigated or no longer maintained, the protocol can simply resume execution, at which point the identity of the user will be revealed. Since the committee members are revealed to the user during credential issuance, one can also imagine DoS attacks targeting only the committee members by a corrupt user utilizing this knowledge. However, while such an attack is cheaper to mount, it is not feasible to maintain it indefinitely. Thus, DoS attacks can delay, but not prevent privacy revocations.

Sybil Attacks

Sybil attacks, where a single party poses a multiple parties, are prevented due to the fact that each user needs to enroll (*i.e.*, post to $\mathcal{F}_{PKI}$) in the system with a public key linked to their real identity. Thus we obtain a list of the actual users in the system, preventing Sybil attacks.

5.5.5 Towards an Efficient Instantiation of PAPR Credentials

We here provide a list of building blocks that may be used to efficiently instantiate our $\prod_{PC}$ protocol.

- To prove correct shuffling of committee candidates' public keys, the Bayer and Groth's scheme [23] may be used, and the computational complexity for the prover is $O(m \log(\sqrt{m}))$, where m is the number of committee candidates.
- For Sig, Boneh Boyen signatures may be used [38, Section 4.3], where the computational complexity is constant for both signing and verifying.
- For SPSig, Abe et al.'s scheme $\mathsf{SIG1}$ in [3, Section 4.1] may be used, where the complexity is linear in the size of the message, which in our case makes it constant since in our protocol we only sign single group elements.
- For Enc and C, ElGamal encryption may be used, in the second case we see ciphertexts as commitments and rely on the schemes' binding property.
- Protocols realizing the functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ can be found in [64, 143, 61, 115], respectively.

As described at a high level in Figure 5.4, ZK_{esc}, which is at the core of our protocol, proves the following.

- ① ZK_{ID} states that the user is the owner of $\mathsf{pk}_{\mathcal{P}}$, *i.e* it knows the secret key $\mathsf{sk}_{\mathcal{P}}$, *and* knows a signature generated by the issuer on $\mathsf{pk}_{\mathcal{P}}$, *i.e* $\sigma_E(\mathsf{pk}_{\mathcal{P}})$. Thus the computational complexity to prove it is constant $\mathcal{O}(1)$.
- ② ZK_{share} states that (2.1) the n shares are constructed correctly, *i.e.*, any set of k shares will reconstruct to the users public key $\mathsf{pk}_{\mathcal{P}}$. Further, (2.2) each of these shares, (2.3) is correctly encrypted, (2.4) for the correct committee member, (2.5) which is correctly indicated in $\vec{q}$. Each of these steps introduces a computational complexity that is linear with respect to n.

The overall complexity of ZK_{esc} is therefore $\mathcal{O}(n)$. Moreover, to instantiate $\prod_{PC}$ efficiently, but without Universal Composability, the ideal functionalities $\mathcal{F}_{BB}$, $\mathcal{F}_{PKI}$, $\mathcal{F}_{ZK}$ and $\mathcal{F}_{NIZK}$ may be heuristically substituted respectively by a blockchain such as Ethereum (note that $\mathcal{F}_{BB}$ may also be implemented starting

from consensus protocols such as those in [69, 126, 84, 79, 11, 12, 138, 177]), a PKI with key transparency such as CONIKS [146], Schnorr proofs over the Tor network and Groth-Sahai proofs [116]. We stress that the security of these substitutions would be heuristic. If formally proven secure, the resulting scheme would at best be proven *sequentially* composable, due to the nature of Groth-Sahai proofs.

In such a system where $\mathcal{F}_{NIZK}$ is substituted for Groth-Sahai proofs, we note that parts (2.3) and (2.4) of ZK_{esc} in Figure 5.4 corresponds to a PVSS scheme. Thus, they can be realized as the verification equations of a pairing-based PVSS scheme, *e.g.*, [63].

Chapter 6

Conclusion

In this thesis, we explored the area lying between privacy-preserving computation and blockchain applications. In particular, we considered auctions, decentralized finance (DeFi) and anonymous credentials.

In the context of auctions, we proposed efficient MPC protocols, *i.e.*, with no need of a trusted third party, for both first and second-price sealed-bid auctions with fairness against rational adversaries, leveraging secret cryptocurrency transactions and public smart contracts. In our approach, it is ensured that cheaters are *identified* and *financially punished* by losing a *secret collateral deposit*, which represents a novel technique. As a *future work*, it may be extended to other contexts where a public deposit is not suitable to achieve fairness.

In the context of decentralized finance, we have proposed a schema of front-running mitigation categories, assessed state-of-the-art techniques in each category and illustrated remaining attacks. Given the financial loss and network congestion resulting from front-running in practice, this thesis highlights the need to develop as *future work* efficient protocols which realize such mitigation techniques.

Finally, in the context of anonymous credentials, we have introduced the notion of anonymous credentials with *Publicly Auditable Privacy Revocation* (PAPR), formalized it as an ideal functionality and proposes a realization that is secure under standard assumptions in the Universal Composability (UC) framework. Keep studying efficient non-UC instantiations of PAPR is left as a *future work*. In particular, PAPR credentials simultaneously provide *conditional* user privacy and *auditable* privacy revocation. Interestingly, the second property enriches anonymous credential systems with transparency by design, effectively discouraging the usage of such systems for mass surveillance.

Indeed, the ultimate goal of this thesis has been to contribute to show how cryptography, and in this specific case blockchain and privacy-preserving computation, can benefit society and the people by reducing the power and the amount of privacy that currently relies on trusted third parties in the real world.

Bibliography

[1] et al. Abadi, M. An open letter from us researchers in cryptography and information security, Jan 2014.

[2] Michel Abdalla, Chanathip Namprempre, and Gregory Neven. On the (im)possibility of blind message authentication codes. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 262–279, San Jose, CA, USA, February 13–17, 2006. Springer, Heidelberg, Germany.

[3] Masayuki Abe, Melissa Chase, Bernardo David, Markulf Kohlweiss, Ryo Nishimaki, and Miyako Ohkubo. Constant-size structure-preserving signatures: Generic constructions and simple assumptions. *Journal of Cryptology*, 29(4):833–878, October 2016.

[4] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *Journal of Cryptology*, 29(2):363–421, April 2016.

[5] Masayuki Abe and Koutarou Suzuki. M+1-st price auction using homomorphic encryption. In David Naccache and Pascal Paillier, editors, *PKC 2002: 5th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 115–124, Paris, France, February 12–14, 2002. Springer, Heidelberg, Germany.

[6] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder–Scalable, Robust Anonymous Committed Broadcast. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1233–1252, 2020.

[7] Mohammad Akbarpour and Shengwu Li. Credible auctions: A trilemma. *Econometrica*, 88(2):425–467, 2020.

[8] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[9] Guillermo Angeris, Alex Evans, and Tarun Chitra. A Note on Privacy in Constant Function Market Makers. *arXiv preprint arXiv:2103.01193*, 2021. https://arxiv.org/abs/2103.01193.

[10] Avalanche. Apricot Phase Four: Snowman++ and Reduced C-Chain Transaction Fees. https://medium.com/avalancheavax/apricot-phase-four-snowman-and-reduced-c-chain-transaction-fees-1e1f67b42ecf, 2021.

[11] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 913–930, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[12] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[13] Samiran Bag, Feng Hao, Siamak F Shahandashti, and Indranil G Ray. Seal: Sealed-bid auction without auctioneers. *IEEE Transactions on Information Forensics and Security*, 2019.

[14] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 1087–1098, Berlin, Germany, November 4–8, 2013. ACM Press.

[15] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of Automated Market Makers in DeFi. In *International Conference on Coordination Languages and Models*, pages 168–187. Springer, 2021. https://doi.org/10.1007/978-3-030-78142-2_11.

[16] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing Extractable Value from Automated Market Makers. *arXiv preprint arXiv:2106.01870*, 2021. To appear in FC'22. https://arxiv.org/pdf/2106.01870.

[17] Olivier Baudron and Jacques Stern. Non-interactive private auctions. In Paul F. Syverson, editor, *FC 2001: 5th International Conference on Financial Cryptography*, volume 2339 of *Lecture Notes in Computer Science*, pages 364–378, Grand Cayman, British West Indies, February 19–22, 2002. Springer, Heidelberg, Germany.

[18] Carsten Baum, Bernardo David, and Rafael Dowsley. A framework for universally composable publicly verifiable cryptographic protocols. *IACR Cryptol. ePrint Arch.*, 2020:207, 2020.

[19] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: Efficient secure computation with financial penalties. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 404–420, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Heidelberg, Germany.

[20] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured MPC: Efficient secure computation with financial penalties. In *International Conference on Financial Cryptography and Data Security*, pages 404–420. Springer, 2020.

[21] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2DEX: privacy-preserving decentralized cryptocurrency exchange. In *International Conference on Applied Cryptography and Network Security*, pages 163–194. Springer, 2021.

[22] Carsten Baum, James Hsin yu Chiang, Bernardo David, Tore Kasper Frederiksen, and Lorenzo Gentile. SoK: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Report 2021/1628, 2021. https://eprint.iacr.org/2021/1628 (to appear in Financial Cryptography and Data Security, FC 2022 International Workshops, DeFi'22).

[23] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[24] Zuzana Beerliova-Trubiniova and Martin Hirt. Efficient multi-party computation with dispute control. In *Theory of Cryptography Conference*, pages 305–328. Springer, 2006.

[25] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 356–374, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

[26] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

[27] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[28] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Annual cryptology conference*, pages 90–108. Springer, 2013.

[29] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[30] F. Benhamouda, S. Halevi, and T. Halevi. Supporting private data on hyperledger fabric with secure multiparty computation. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 357–363, April 2018.

[31] 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: 18th Theory of Cryptography Conference, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 260–290, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

[32] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[33] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 410–440, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

[34] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012.

[35] Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference*

on Applied Cryptography and Network Security, volume 10892 of *Lecture Notes in Computer Science*, pages 221–239, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.

[36] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009: 13th International Conference on Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, Accra Beach, Barbados, February 23–26, 2009. Springer, Heidelberg, Germany.

[37] Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006: 10th International Conference on Financial Cryptography and Data Security*, volume 4107 of *Lecture Notes in Computer Science*, pages 142–147, Anguilla, British West Indies, February 27 – March 2, 2006. Springer, Heidelberg, Germany.

[38] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[39] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

[40] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[41] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany.

[42] Felix Brandt. Secure and private auctions without auctioneers. *Technical Report FKI-245–02. Institut fur Informatick, Technishce Universitat Munchen*, 2002.

[43] Felix Brandt. Fully private auctions in a constant number of rounds. In Rebecca Wright, editor, *FC 2003: 7th International Conference on Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages

223–238, Guadeloupe, French West Indies, January 27–30, 2003. Springer, Heidelberg, Germany.

[44] Felix Brandt. How to obtain full privacy in auctions. *International Journal of Information Security*, 5(4):201–216, 2006.

[45] Felix Brandt and Tuomas Sandholm. Efficient privacy-preserving protocols for multi-unit auctions. In Andrew Patrick and Moti Yung, editors, *FC 2005: 9th International Conference on Financial Cryptography and Data Security*, volume 3570 of *Lecture Notes in Computer Science*, pages 298–312, Roseau, The Commonwealth Of Dominica, February 28 – March 3, 2005. Springer, Heidelberg, Germany.

[46] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1335–1352, Baltimore, MD, August 2018. USENIX Association.

[47] Joakim Brorsson, Bernardo David, Lorenzo Gentile, Elena Pagnin, and Paul Stankovski Wagner. PAPR: Publicly auditable privacy revocation for anonymous credentials, 2023. (to appear in CT-RSA 2023, Cryptographers' Track at RSA Conference).

[48] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[49] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[50] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–326. Springer, 2021. https://doi.org/10.1007/978-3-030-77870-5_11.

[51] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. Practical UC-secure delegatable credentials with attributes and their application to blockchain. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 683–699, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[52] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In Tetsu Iwata and Jung Hee

Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 262–288, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[53] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 201–210, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[54] Jan Camenisch and Anja Lehmann. (Un)linkable pseudonyms for governmental databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 1467–1479, Denver, CO, USA, October 12–16, 2015. ACM Press.

[55] Jan Camenisch and Anja Lehmann. Privacy-preserving user-auditable pseudonym systems. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 269–284. IEEE, 2017.

[56] Jan Camenisch and Anna Lysyanskaya. An efficient system for nontransferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[57] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[58] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical Report/ETH Zurich, Department of Computer Science*, 260, 1997.

[59] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[60] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[61] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140, 2002. https://eprint.iacr.org/2002/140.

[62] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[63] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556, Kanazawa, Japan, July 10–12, 2017. Springer, Heidelberg, Germany.

[64] Ignacio Cascudo and Bernardo David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 311–341, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[65] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint Archive, Report 2022/242, 2022. https://eprint.iacr.org/2022/242.

[66] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.

[67] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[68] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

[69] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[70] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[71] Tarun Chitra, Guillermo Angeris, and Alex Evans. Differential Privacy in Constant Function Market Makers. *Cryptology ePrint Archive*, 2021. https://eprint.iacr.org/2021/1101.

[72] 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: 24th Conference on Computer and Communications Security*, pages 719–728, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[73] Shumo Chu, Yu Xia, and Zhenfei Zhang. Manta: a Plug and Play Private DeFi Stack. 2021. https://eprint.iacr.org/2021/743.

[74] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing*, pages 364–369, Berkeley, CA, USA, May 28–30, 1986. ACM Press.

[75] Peter Cramton et al. Spectrum auctions. *Handbook of telecommunications economics*, 1:605–639, 2002.

[76] Elizabeth C. Crites and Anna Lysyanskaya. Delegatable anonymous credentials from mercurial signatures. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 535–555, San Francisco, CA, USA, March 4–8, 2019. Springer, Heidelberg, Germany.

[77] Mariana Botelho da Gama, John Cartlidge, Antigoni Polychroniadou, Nigel P. Smart, and Younes Talibi Alaoui. Kicking-the-Bucket: Fast Privacy-Preserving Trading Using Buckets. Cryptology ePrint Archive, Report 2021/1549, 2021. To appear in FC'22, https://ia.cr/2021/1549.

[78] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *IEEE Symposium on Security and Privacy*, pages 910–927. IEEE, 2020.

[79] 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: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany.

[80] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer*

Science, pages 572–590, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

[81] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

[82] Vincent Danos, Hamza El Khalloufi, and Julien Prat. Global Order Routing on Exchange Networks. In *International Conference on Financial Cryptography and Data Security*, pages 207–226. Springer, 2021.

[83] Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018: 22nd International Conference on Financial Cryptography and Data Security*, volume 10957 of *Lecture Notes in Computer Science*, pages 500–519, Nieuwpoort, Curaçao, February 26 – March 2, 2018. Springer, Heidelberg, Germany.

[84] 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, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[85] Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. FAST: Fair auctions via secret transactions. Cryptology ePrint Archive, Report 2021/264, 2021. https://eprint.iacr.org/2021/264.

[86] Bernardo David, Lorenzo Gentile, and Mohsen Pourpouneh. FAST: Fair auctions via secret transactions. In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 22: 20th International Conference on Applied Cryptography and Network Security*, volume 13269 of *Lecture Notes in Computer Science*, pages 727–747, Rome, Italy, June 20–23, 2022. Springer, Heidelberg, Germany.

[87] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. GearBox: An efficient UC sharded ledger leveraging the safety-liveness dichotomy. Cryptology ePrint Archive, Report 2021/211, 2021. https://eprint.iacr.org/2021/211.

[88] Vanesa Daza, Abida Haque, Alessandra Scafuro, Alexandros Zacharakis, and Arantxa Zapico. Mutual accountability layer: accountable anonymity within accountable trust. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 318–336. Springer, 2022.

[89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[90] Dominic Deuber, Nico Döttling, Bernardo Magri, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Minting mechanism for proof of stake blockchains. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20: 18th International Conference on Applied Cryptography and Network Security, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 315–334, Rome, Italy, October 19–22, 2020. Springer, Heidelberg, Germany.

[91] Jannik Dreier, Jean-Guillaume Dumas, and Pascal Lafourcade. Brandt's fully private auction protocol revisited. *Journal of Computer Security*, 23(5):587–610, 2015.

[92] Andreas Erwig, Sebastian Faust, and Siavash Riahi. Large-scale non-interactive threshold cryptosystems through anonymity. Cryptology ePrint Archive, Report 2021/1290, 2021. https://eprint.iacr.org/2021/1290.

[93] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In *Financial Cryptography*, pages 170–189, Cham, 2020. Springer International Publishing.

[94] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[95] 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.

[96] Jonathan Frankle, Sunoo Park, Daniel Shaar, Shafi Goldwasser, and Daniel J. Weitzner. Practical accountability of secret processes. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 657–674, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[97] Matthew K Franklin and Michael K Reiter. The design and implementation of a secure auction service. *IEEE Transactions on Software Engineering*, 22(5):302–312, 1996.

[98] Hisham S. Galal and Amr M. Youssef. Trustee: Full privacy preserving vickrey auction on top of Ethereum. In Andrea Bracciali, Jeremy Clark,

Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *Lecture Notes in Computer Science*, pages 190–207, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Heidelberg, Germany.

[99] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[100] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006.

[101] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[102] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[103] 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, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 64–93, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[104] 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: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 32–61, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

[105] 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.

[106] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[107] 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 Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[108] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[109] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[110] Shafi Goldwasser and Sunoo Park. Public accountability vs. secret laws: Can they coexist? a cryptographic proposal. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, pages 99–110, 2017.

[111] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, 2022.

[112] Matthew Green, Gabriel Kaptchuk, and Gijs Van Laer. Abuse resistant law enforcement access systems. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 553–583, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

[113] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 321–340, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

[114] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016.

[115] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of the ACM (JACM)*, 59(3):1–35, 2012.

[116] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[117] Feng Hao and Piotr Zieliński. A 2-round anonymous veto protocol. In *International Workshop on Security Protocols*, pages 202–211. Springer, 2006.

[118] Michael Harkavy, J Doug Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In *USENIX Workshop on Electronic Commerce*, 1998.

[119] Martin Hellman. Open letter to Senator Ron Wyden, Feb 2018.

[120] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[121] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[122] Ari Juels and Michael Szydlo. A two-server, sealed-bid auction protocol. In Matt Blaze, editor, *FC 2002: 6th International Conference on Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 72–86, Southampton, Bermuda, March 11–14, 2003. Springer, Heidelberg, Germany.

[123] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-Fair Consensus in the Permissionless Setting. *IACR Cryptol. ePrint Arch.*, 2021:139, 2021. https://eprint.iacr.org/2021/139.

[124] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, Strong Order-Fairness in Byzantine Consensus. *Cryptology ePrint Archive*, 2021. https://eprint.iacr.org/2021/1465.

[125] Rami Khalil, Arthur Gervais, and Guillaume Felley. Tex-a securely scalable trustless exchange. *Cryptology ePrint Archive*, 2019. https://eprint.iacr.org/2019/265.

[126] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[127] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 705–734, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[128] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.

[129] Paul Klemperer. *Auctions: theory and practice*. Princeton University Press, 2004.

[130] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *2021 IEEE Symposium on Security and Privacy*, pages 608–625, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[131] Vijay Krishna. *Auction theory*. Academic press, 2009.

[132] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 418–429, Vienna, Austria, October 24–28, 2016. ACM Press.

[133] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 195–206, Denver, CO, USA, October 12–16, 2015. ACM Press.

[134] Ranjit Kumaresan, Vinod Vaikuntanathan, and Prashant Nalini Vasudevan. Improvements to secure computation with penalties. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 406–417, Vienna, Austria, October 24–28, 2016. ACM Press.

[135] Kaoru Kurosawa and Wakaha Ogata. Bit-slice auction circuit. In *European Symposium on Research in Computer Security*, pages 24–38. Springer, 2002.

[136] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 25–36, 2020.

[137] Yunqi Li. HoneyBadgerSwap: Making MPC as a Sidechain. https://medium.com/initc3org/honeybadgerswap-making-mpc-as-a-sidechain-364bebdb10a5, 2021.

[138] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. In *34th Annual ACM Symposium on Theory of Computing*, pages 514–523, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[139] Helger Lipmaa, N. Asokan, and Valtteri Niemi. Secure Vickrey auctions without threshold trust. In Matt Blaze, editor, *FC 2002: 6th International Conference on Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 87–101, Southampton, Bermuda, March 11–14, 2003. Springer, Heidelberg, Germany.

[140] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 887–903, 2019.

[141] Wouter Lueks, Maarten H Everts, and Jaap-Henk Hoepman. Vote to link: Recovering From Misbehaving Anonymous Users. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, pages 111–122, 2016.

[142] Andreu Mas-Colell, Michael Dennis Whinston, Jerry R Green, et al. *Microeconomic theory*, volume 1. Oxford university press New York, 1995.

[143] Daniel Masny and Gaven J. Watson. A PKI-based framework for establishing efficient MPC channels. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 1961–1980, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[144] Greg Maxwell. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt, 2016.

[145] Greg Maxwell. Confidential transactions. https://people.xiph.org/greg/confidential_values.txt,, 2016.

[146] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015: 24th USENIX Security Symposium*, pages 383–398, Washington, DC, USA, August 12–14, 2015. USENIX Association.

[147] Silvio Micali. Fair Cryptosystems. Technical report, Massachusetts Institute of Technology, 1993.

[148] Peter Bro Miltersen, Jesper Buus Nielsen, and Nikos Triandopoulos. Privacy-enhancing auctions using rational cryptography. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 541–558, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Heidelberg, Germany.

[149] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[150] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)*, 2008.

[151] Hannu Nurmi and Arto Salomaa. Cryptographic protocols for vickrey auctions. *Group Decision and Negotiation*, 2(4):363–373, 1993.

[152] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 80–99, New York, NY, USA, March 4–7, 2006. Springer, Heidelberg, Germany.

[153] Paradigm. Ethereum is a Dark Forest. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/, 2020.

[154] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE, 2013.

[155] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[156] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer, Heidelberg, Germany.

[157] Penumbra. ZSwap documentation. https://protocol.penumbra.zone/main/zswap.html, 2021.

[158] Daniel Perez, Sam M Werner, Jiahua Xu, and Benjamin Livshits. Liquidations: DeFi on a Knife-edge. In *International Conference on Financial Cryptography and Data Security*, pages 457–476. Springer, 2021.

[159] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[160] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-locked Puzzles and Time-release Crypto. https://people.csail.mit.edu/rivest/pubs/RSW96.pdf, 1996.

[161] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[162] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[163] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981.

[164] Shutter. Shutter Network. https://shutter.network/, 2022.

[165] Markus Stadler. *Cryptographic protocols for revocable privacy*. PhD thesis, Verlag nicht ermittelbar, 1996.

[166] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology – EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219, Saint-Malo, France, May 21–25, 1995. Springer, Heidelberg, Germany.

[167] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-Times anonymous authentication (extended abstract). In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 308–322, Jeju Island, Korea, December 5–9, 2004. Springer, Heidelberg, Germany.

[168] et al. Urs, Gasser. Don't panic: Making progress on the "going dark " debate, Feb 2016.

[169] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 991–1008, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[170] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[171] Ye Wang, Yan Chen, Shuiguang Deng, and Roger Wattenhofer. Cyclic Arbitrage in Decentralized Exchange Markets. *Available at SSRN 3834535*, 2021. https://dx.doi.org/10.2139/ssrn.3834535.

[172] Dave White, Dan Robinson, and Hayden Adams. Time-weighted Average Market Maker (TWAMM). 2021. https://www.paradigm.xyz/2021/07/twamm/.

[173] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[174] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[175] Andrew Chi-Chih Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[176] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[177] 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 Symposium Annual on Principles of Distributed Computing*, pages 347–356, Toronto, ON, Canada, July 29 – August 2, 2019. Association for Computing Machinery.

[178] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. *arXiv preprint arXiv:2103.02228*, 2021. https://arxiv.org/abs/2103.02228.