THE UNIVERSITY
of EDINBURGH

THE UNIVERSITY
of EDINBURGH

SECURITY AND PRIVACY OF INCENTIVE-DRIVEN MECHANISMS

Yun Lu

A thesis submitted for the degree of
Doctor of Philosophy

The University of Edinburgh
School of Informatics

2022

Acknowledgments

I have received great mentorship and advice throughout my PhD studies and the writing of this dissertation.

I would like to first thank my wonderful supervisor Professor Vassilis Zikas for his many years of support and care both in my research and in my academic career. Your invaluable guidance has made me a better writer and researcher.

I would also like to thank my mentors and colleagues from the University of Edinburgh, Rensselaer Polytechnic Institute, and UCLA, who generously shared their experiences and expertise with me. I would especially like to single out my annual review panel members, Professors Aggelos Kiayias, Kousha Etessami, and Markulf Kohlweiss for their advice and encouragement.

I would also like to thank my exceptional coauthors: Christian Badertscher, Michele Ciampi, Ao Liu, and Professor Lirong Xia. Thank you for the countless hours of discussions and all those late-night writing sessions before a deadline.

Last but not least, I would like to thank my parents for your unconditional love and understanding. Many thanks for the support from my amazing friends and family, and your patience with my often strange working hours.

Abstract

While cryptographic tools offer practical security and privacy supported by theory and formal proofs, there are often gaps between the theory and intricacies of the real world. This is especially apparent in the realm of *game theoretic applications* where protocol participants are motivated by incentives and preferences on the protocol outcome. These incentives can lead to additional requirements or unexpected attack vectors, making standard cryptographic concepts inapplicable.

The goal of this thesis is to bridge some of the gaps between cryptography and incentive-driven mechanisms. The thesis will consist of three main research threads, each studying the privacy or security of a game-theoretic scenario in non-standard cryptographic frameworks in order to satisfy the scenario's unique requirements. Our first scenario is preference aggregation, where we will analyze the privacy of voting rules while requiring the rules to be deterministic. Then, we will study games, and how to achieve collusion-freeness (and its composable version, collusion-preservation) in the decentralized setting. Finally, we explore the robustness of Nakamoto-style proof-of-work blockchains against 51% attacks when the main security assumption of honest majority fails. Most of the results in this thesis are also published in the following (in order): Ch. 3: [103], Ch. 4: [47], and Ch. 5: [104].

Our first focus is preference aggregation—in particular voting rules. Specifically, we answer the crucial question: *How private is the voting rule we use and the voting information we release?* This natural and seemingly simple question was sidestepped in previous works, where randomization was added to voting rules in order to achieve the widely-known notion of *differential privacy* (DP). Yet, randomness in an election can be undesirable, and may alter voter incentives and strategies. In this chapter of our thesis, we expand and improve upon previous works and study deterministic voting rules. In a similarly well-accepted framework of *distributional differential privacy* (DDP), we develop new techniques in analyzing and comparing the privacy of voting rules—leading to a new measure to contrast different rules in addition to existing ones in the field of social choice. We learn the positive message that even vote tallies have very limited privacy leakage that decreases quickly in the number of votes, and a surprising fact that outputting the winner using different voting rules can result in asymptotically different privacy leakage.

Having studied privacy in the context of parties with preferences and incentives, we turn our attention to the secure implementation of games. Specifically, we study the issue of collusion and how to avoid it. Collusion, or subliminal communication, can introduce undesirable coalitions in games that allow malicious parties, e.g. cheating poker players, a wider set of strategies. Standard cryptographic security is insufficient to address the issue, spurring on a line of work that defined and constructed *collusion-free* (CF), or its composable version, *collusion-preserving* (CP) protocols. Unfortunately, they all required strong assumptions on the communication medium, such as physical presence of the parties, or a restrictive star-topology network with a trusted

mediator in the center. In fact, CF is impossible without restricted communication, and CP is conjectured to always require a mediator. Thus, circumventing these impossibilities is necessary to truly implement games in a decentralized setting. Fortunately, in the rational setting, the attacker can also be assumed to have utility. By ensuring collusion is only possible by sending incorrect, penalizable messages, and composing our protocol with a blockchain protocol as the source of the penalization, we prove our protocol as CP against incentive-driven attackers in a framework of rational cryptography called *rational protocol design* (RPD).

Lastly, it is also useful to analyze the security of the blockchain and its associated cryptocurrencies—cryptographic transaction ledger protocols with embedded monetary value—using a rational cryptography framework like RPD. Our last chapter studies the incentives of attackers that perform *51% attacks* by breaking the main security assumption of *honest majority* in proof-of-work (PoW) blockchains such as Bitcoin and Ethereum Classic. Previous works abstracted the blockchain protocol and the attacker's actions, analyzing 51% attacks via various techniques in economics or probability theory. This leads open the question of exploring this attack in a model closer to standard cryptographic analyses. We answer this question by working in the RPD framework. Improving upon previous analyses that geared towards only mining rewards, we construct utility functions that model the incentives of 51% attackers. Under the RPD framework, we are able to determine when an attacker is incentivized to attack a given instantiation of the blockchain protocol. More importantly, we can make general statements that indicate changes to protocol parameters to make it secure against all rational attackers under these incentives.

Contents

Chapter 1

Introduction

When everyday internet users think about the word *cryptography*, what image comes to their mind? Is it a Caesar cipher wheel? Encryption standards such as AES, the TLS protocol which provides security for browsing the internet, or hash functions like SHA1?

While the above are all example of cryptography, the field has expanded far beyond secret messages or encrypted passwords. As more individual data and everyday processes move online, users are becoming more aware that cryptography has now been tasked with protecting their privacy, assets, and ensuring the integrity of systems. Even traditionally in-person activities, such as games, have also grown its online presence. In games like online poker where real money is involved, encryption may not be sufficient. Implementing a fair game involves, for example, constructing a fair card shuffling mechanism, a system to deter cheating, and a method to distribute the players' winnings. These winnings may even be in a cryptocurrency, another (perhaps nowadays most well-known) application of cryptography. Blockchains protocols, which underlie most cryptocurrencies, usually involve a myriad of cryptographic primitives like hash and pseudorandom functions.

Constraints on protocols with incentives In the above examples, the parties involved are often not just concerned about their privacy or the security of the system, but they also have *incentives* on the outcome of a process. For example, surveyees may want the census to result in policies reflecting their responses, online poker players want to win the game, and cryptocurrency users want more coins.

Yet, most common models in cryptographic analyses do not consider incentives or the strategies that parties adopt as a result. In fact, in most adversary models, there are only two types of parties: *honest* and *corrupt*. The honest parties always act according to the protocol specification, and corrupt parties are controlled by a (malicious) attacker who may instruct them to act in some arbitrary manner (under constraints of their capabilities e.g., as polynomial-time interactive Turing machines). If corrupt parties in a protocol can somehow break the desired security definition, even by acting clearly against their own interest, we say the protocol is *insecure*, and otherwise it is considered *secure*.

Even the term *fairness* may not mean the same thing to cryptographers as everyday users. In cryptography, fair protocols are those where misbehaving parties cannot receive the protocol output without honest parties *also* receiving those outputs. In standard English, however, being *fair* often refers to the more general concept that no party gains an undeserved amount of reward over others—a concept which only makes sense when considering incentives. This distinction means that even protocols with cryptographic fairness may not in fact seem fair to its participants. Here, we point out three properties which are important for "fairness", in the standard English sense, but which cryptographically secure or private mechanisms can fail:

1. The protocol outcome should be correct based on the parties' inputs
2. The protocol should uphold the rules of the process (e.g., game) it is supposed to implement.
3. The protocol should incentivize honest behavior.

Below, we give an overview of how standard cryptographic frameworks can fail each of these properties, how these frameworks and models can be altered to achieve them, and even how to use incentives to deter malicious behavior.

Deterministic voting rules The first constraint we explore is determinism, and the property that **the protocol outcome should be correct based on the parties' inputs**. In cases where the parties have biases over the protocol outcome, such as in preference aggregation mechanisms, the lack of determinism can lead to parties changing their strategies, and they might not even believe and may dispute the outcome. A prominent example is the 2016 Iowa caucus in the United States, where county delegates in several Iowa precincts were awarded based on coin flips. Even after the eventual winner was announced, the impact of such coin flips on the result was still controversial [48].

Unfortunately, determinism is incompatible with *differential privacy* (DP), which as we recall is the privacy technique trusted to protect individual responses in the 2020 US Census. In order to achieve DP, a data release mechanism such as tallies of survey responses (or, in the case of the Iowa caucus, the winner of an election) must ensure outcome has a low probability to be decided based on one individual's input. This inclusion of probabilities means that the data release mechanism must be randomized. Indeed, previous works on the privacy of preference aggregation mechanisms used randomization to achieve privacy [84, 98, 121]. Whereas this created private mechanisms that have DP, these techniques say nothing about the privacy of deterministic mechanisms, such as most common voting rules. Fortunately, this restriction with DP has been observed in previous works (though not in terms of preference aggregation), and various alternative privacy definitions based on DP exist. In our thesis, we will use *distributional differential privacy* (DDP) [21, 79] to study the privacy of deterministic voting rules. We choose DDP due to its generality, and assuming independently distributed votes (e.g., the common simplifying probabilistic model in social choice called *impartial culture*), the DDP definition can

be simplified to more easily compare the privacy of voting rules.

In Ch. 3, we quantify the DDP parameters of voting rules with respect to a variety of vote distributions. We improve upon an existing privacy analysis of the histogram function, and rank the privacy in a class of voting rules called the *generalized scoring rules* (GSR) [127]. We learn that for most voting rules, as long as the probability of ties in the election result is not too high, the privacy loss is exponentially small in the number of votes, and within GSR we show that there is asymptotic separation of privacy among different voting rules.

Collusion in games The inclusion of randomness is not only an issue in voting, but it can also cause trouble when implementing games. In particular, it exacerbates the problem of *collusion*, and breaks the property that **the protocol should uphold the rules of the process it is supposed to implement**. In games where parties are supposed to keep some information secret to themselves (e.g., the cards they are dealt in a card game), colluding parties who work together (usually against the rules) can gain a big advantage over other players. In online poker for example, collusion is a major concern. To detect such behavior, suspicious cases are generally manually verified by experts, and by running certain action-tracking software during the game [5, 7]. This is of course not a definite way to prevent or even detect collusion. For example, if parties have an undetectable private communication channel among themselves, such as private phone calls between players, then collusion is unavoidable. Private channels are not a requirement for collusion, however. Since the usual security definition itself does not consider the problem of collusion, and instead implicitly assumes all malicious parties as already colluding (by being controlled by a single adversary), parties can still collude in secure implementations of games. In fact, secure protocols can even be a detriment to detecting collusion. In such protocols, messages (e.g. encryptions or signatures) usually are randomized, allowing parties to hide secret colluding messages inside them.

The works of Alwen et al. and Lepinski et al. [8, 11, 100] formally defined *collusion-freeness* (CF), and in Alwen et al. [9] its composable version[1], *collusion-preservation* (CP). Informally, a protocol is collusion-free if the participating parties can only learn as much information from participating in the protocol, as they would in an ideal scenario where collusion is disallowed. For example, in a poker game, if two parties collude by revealing their hand to each other, this would break this definition of collusion-freeness, since in the ideal non-colluding scenario the players would not know each other's cards. Alwen et al. [8, 9, 11] introduced a solution in the "mediated model", where parties are only allowed to communicate with the trusted central party called the mediator. However, the collusion-freeness and preservation of this solution heavily relies on the honesty of the mediator. As we also show, when the mediator is dishonest, desirable properties like being able to identify a cheating party, or even knowing

[1] "Composable" here refers to the definition of collusion-freeness not only being satisfied when a game is running as a standalone protocol, but also in presence of other cryptographic protocols. In particular, supposing parties exchange a limited number of bits (e.g., secret keys) via another protocol, this should not lead to an exponential blowup in the collusion in our game protocol.

whether a game has ended, are also lost. Moreover, this centralized solution does not fit with the decentralized nature of the internet, such as playing games over the blockchain [53, 54]. On the other hand, it seemed like restrictive models like the mediated model is necessary to achieve CF/CP. Unlike the mediated model where the mediator can filter invalid messages, it is easy to show that attackers can technically always "collude" by sending invalid colluding messages (e.g., containing the cards they are holding), if given unfiltered communication channels.

In Ch. 4, we construct a CP protocol[2] using semi-trusted hardware tokens, which allow parties to play games over an authenticated (but unfiltered) broadcast network, while eliminating the issues of the mediated model. To deter attackers from abusing the unfiltered nature of the broadcast channel, we augment our solution with a penalization scheme, which can be implemented on e.g., the blockchain and imposes a financial penalization to cheating. Assuming rational strategies and sufficient penalties, we can also show that honest strategy is the dominant strategy. We prove this in a framework called *rational protocol design* (RPD) [65], which augments the standard cryptographic security models by incorporating incentives. Instead of showing that the protocol is secure against all malicious attackers, RPD relaxes the security definition by considering protocols as secure, as long as attackers cannot earn an "undeserved" amount of utility from attacking the protocol. In our case, we alter the framework for the setting of collusion-freeness/preservation, and show that if attackers try to send colluding messages, they will be caught as cheating, and lose a predetermined amount of collateral in the protocol. Lastly, we show that our construction satisfies incentive compatibility, which says that honest behavior is an equilibrium strategy for all parties in the protocol.

51% attacks on blockchains Recall that in Ch. 4, one way to deter malicious behavior is to implement a penalization scheme, which can be realized through a blockchain whose associated cryptocurrency has monetary value. In the last chapter of the thesis, we demonstrate how, as the blockchain itself is a cryptographic protocol with inherent incentives, these incentives can conflict with analyses in standard cryptographic frameworks. In particular, it can break the property that **the protocol should incentivize honest behavior**.

The security analysis in standard blockchain literature [17, 66, 115] uses the usual model that parties are either honest corrupt. Corrupt parties may act arbitrarily and honest parties always act according to the protocol specification, both irrespective of incentives. In this setting, blockchain is proven secure assuming that the majority of mining power (e.g., in proof-of-work (PoW) systems e.g., Bitcoin or Ethereum Classic this is hashing power, and in proof-of-stake (PoS) e.g., Algorand, Ouroboros, and Snow White this is the fraction of owned stake) is honest. Assuming honest majority (and appropriate protocol parameters are selected according to the total mining power in the system), the main security property, *consistency* is achieved. Roughly, consistency says that after a transaction becomes part of a blockchain for a long-enough

[2]As a side note, while collusion-freeness and their solutions has been first defined for games, works have extended its application to *cryptographic reverse firewalls* [108], which prevent outgoing messages of a machine from leaking its secrets (e.g. passwords or keys).

"confirmation time", it becomes immutable and cannot be reverted nor spent again.

In Ch. 5, we focus on "Nakamoto-style" PoW blockchains, which are based on Nakamoto's Bitcoin protocol [111] but possibly with different hashing algorithms. In these blockchain protocols, honest majority was originally deemed a reasonable assumption, with the informal argument that: (1) it is unlikely for any coordinated group of parties to gain a majority of power in the system, and (2) the rewards for successfully mining blocks will incentivize honest mining behavior. Unfortunately, this assumption has been broken repeatedly [1], especially for blockchains with a relatively low total hashing power in the system. In such systems, it is easier to gain a majority of hashing power and break honest majority. This in turn makes consistency impossible to achieve in standard cryptographic analyses: Given a majority of hashing power, an attacker can make blocks that fork the blockchain, tricking different parties into having different views on the chain and making transactions easier to double-spend. The reality of these *51% (double-spending) attacks* spurred a line of works that study them in various rational settings where attacks have cost and rewards, and attackers are incentivized by profit. These analyses generally employ techniques from economic or probability theory, which while making substantial statements on the profitability of attacks, also make non-composable abstractions on the details of the blockchain and limit parties to a prescribed set of actions.

In this last chapter, we study 51% double-spending attacks in the RPD framework, which mitigates the above issues of previous works. Recalling from Ch. 4, RPD is a rational framework that is compatible with cryptographic functionalities. We improve upon the utility function of the previous work [15] on RPD analysis of blockchains by augmenting it with incentives for double-spending attacks, using our improved utility function to show a range of parameters where forking is the dominant strategy. Lastly, our model also allows us to make general statements in the other direction. We show how to amplify a protocol parameter called the cut-off parameter (in essence, this is the confirmation "time" to wait until a transaction is deemed irreversible and not able to be double-spent), so that honest mining is again the equilibrium solution.

Summary We have seen how incentives impose additional constraints on various application scenarios, and how to alter standard cryptographic frameworks to accommodate them. On the other hand, in Ch. 4 and Ch. 5, we have also seen how incentives can be used to circumvent impossibilities in a reasonably relaxed model with rational attackers. Together, our three main chapters demonstrate the importance of considering incentives when analyzing cryptographic protocols, and the power of incentives-augmented frameworks.

Chapter 2

Preliminaries

2.1 Notation

We denote the security parameter by κ and use "||" as concatenation operator (i.e., if a and b are two strings then by $a||b$ we denote the concatenation of a and b). We denote a set of parties by $\mathcal{P}$ and the set of honest parties $\mathcal{H} \subseteq \mathcal{P}$. The set of corrupted parties is $\mathcal{P} \backslash \mathcal{H}$. We denote probabilistic polynomial time as PPT.

We denote a random variable $\vec{X}$ with distribution π as $\vec{X} \sim \pi$. For a finite set Q, $x \xleftarrow{\$} Q$ denotes a sampling of x from Q with uniform distribution. When it is necessary to refer to the randomness r used by and algorithm A we use the following notation: $A(\cdot; r)$. We denote with $[n]$ the set $\{1, \ldots, n\}$, with $\mathbb{F}$ an arbitrary (but fixed) finite field and with $\mathbb{N}$ the set of non-negative integers (natural numbers).

2.1.1 General cryptography concepts

We formally define relevant basic concepts from cryptography.

Negligible functions We say a function $\mathsf{negl}(\cdot)$ is *negligible* if for every positive integer c there is an integer N_c such that for all $x > N_c$, $|\mathsf{negl}(x)| < 1/x^c$. We say $a \overset{\mathsf{negl}}{\leq} b$ for real values a, b to mean that $a \leq b + \mathsf{negl}(\kappa)$ for negligible function $\mathsf{negl}(\cdot)$.

Pseudo-random functions

Definition 1 (Pseudo-Random Function (from [72]))**.** A function $\mathsf{PRF}\colon \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{c\kappa}$ is called a *pseudo-random function* if it satisfies the following properties.
Efficient: For every $k \in \{0,1\}^\kappa$, and every $m \in \{0,1\}^{c\kappa}$, there exists a PPT algorithm to compute $\mathsf{PRF}_k(m) = \mathsf{PRF}(k, m)$.

Indistinguishable from Random: For every PPT $\mathcal{A}$, there exists a negligible function negl, such that for all auxiliary input $z \in \{0,1\}^*$ it holds that:

$$\left| \Pr_{k \xleftarrow{\$} \{0,1\}^\kappa} (\mathcal{A}^{\mathsf{PRF}_k}(z) = 1) - \Pr_{f \xleftarrow{\$} F} (\mathcal{A}^f(z) = 1) \right| < \mathsf{negl}(\kappa)$$

where $F = \{f \colon \{0,1\}^\kappa \to \{0,1\}^{c\kappa}\}$

Encryption

Definition 2 (CPA-secure Symmetric Encryption Scheme (from notes of [73], Definition 6.8))**.** A triple of PPT algorithms $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where $\mathsf{Gen} : 1^\kappa \to \{0,1\}^{\lambda_1(\kappa)}$, $\mathsf{Enc} : \{0,1\}^{\lambda_1(\kappa)} \times \{0,1\}^\kappa \to \{0,1\}^{\lambda_2(\kappa)}$, $\mathsf{Dec} : \{0,1\}^{\lambda_1(\kappa)} \times \{0,1\}^{\lambda_2(\kappa)} \to \{0,1\}^\kappa$ and λ_i are polynomials, is called a *chosen-plaintext-attack-secure symmetric encryption scheme* if it satisfies the following properties.

Completeness: For every secret key $s \xleftarrow{\$} \mathsf{Gen}(1^\kappa)$, and every $m \in \{0,1\}^\kappa$, we have that $\Pr[\mathsf{Dec}(s, \mathsf{Enc}(s, m)) = m] = 1$.

CPA-Security: Let the left-or-right encryption oracle be as follows, where $b \in \{0,1\}$, $m_0, m_1 \in \{0,1\}^\kappa$:

$$\begin{array}{l} \mathsf{Oracle\ Enc}(\mathsf{LR}(m_0, m_1, b)): \\ \quad \mathsf{if}\ |m_0| \neq |m_1|\ \mathsf{then\ return}\ \perp \\ \quad c \xleftarrow{\$} \mathsf{Enc}(m_b) \\ \quad \mathsf{return}\ \ c \end{array}$$

Let $\mathcal{A}$ be an adversary. We consider the following experiments:

$$\begin{array}{l|l} \mathsf{Experiment}\ \mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathsf{SE}}(\mathcal{A}): & \mathsf{Experiment}\ \mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathsf{SE}}(\mathcal{A}): \\ \quad s \xleftarrow{\$} \mathsf{Gen}(1^\kappa) & \quad s \xleftarrow{\$} \mathsf{Gen}(1^\kappa) \\ \quad d \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{LR}(\cdot,\cdot,1))} & \quad d \xleftarrow{\$} \mathcal{A}^{\mathsf{Enc}(\mathsf{LR}(\cdot,\cdot,0))} \\ \quad \mathsf{return}\ d & \quad \mathsf{return}\ d \end{array}$$

For every PPT $\mathcal{A}$, there exists a negligible function negl such that it holds that:

$$\left| \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}1}}_{\mathsf{SE}}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ind\text{-}cpa\text{-}0}}_{\mathsf{SE}}(\mathcal{A}) = 1] \right| < \mathsf{negl}(\kappa)$$

Strong signatures

Definition 3 (Strong unforgeable signature scheme)**.** A triple of PPT algorithms $(\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver})$, where $\mathsf{Kgen} : 1^\kappa \to \{0,1\}^{\lambda_1(\kappa)} \times \{0,1\}^{\lambda_2(\kappa)}$, $\mathsf{Sign} : \{0,1\}^{\lambda_1(\kappa)} \times \{0,1\}^* \to \{0,1\}^{\lambda_3(\kappa)}$, and $\mathsf{Ver} : \{0,1\}^{\lambda_2(\kappa)} \times \{0,1\}^* \times \{0,1\}^{\lambda_3(\kappa)} \to \{0,1\}$, is called a *signature scheme* if it satisfies the following properties.

Completeness: For every pair $(s, v) \xleftarrow{\$} \mathsf{Kgen}(1^\kappa)$, and every $m \in \{0,1\}^\kappa$, we have that $\Pr[\mathsf{Ver}(v, m, \mathsf{Sign}(s, m)) = 0] = 1 - \mathsf{negl}(\kappa)$.

Consistency (non-repudiation): For any m, the probability that $\mathsf{Kgen}(1^\kappa)$ generates (s, v) and $\mathsf{Ver}(v, m, \sigma)$ generates two different outputs in two independent invocations is smaller than $\mathsf{negl}(\kappa)$.

(Strong) Unforgeability: For every PPT $\mathcal{A}$, there exists a negligible function negl, such that for all auxiliary input $z \in \{0,1\}^\star$ it holds that:

$$\Pr[(s, v) \xleftarrow{\$} \mathsf{Kgen}(1^\kappa); (m, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sign}(s,\cdot)}(z, v) \wedge \\ \mathsf{Ver}(v, m, \sigma) = 1 \wedge (m, \sigma) \notin Q] < \mathsf{negl}(\kappa)$$

where Q denotes the set of the couples message-signature $\{(m_i, \sigma_i)\}_{i \in \kappa}$ where m_i is requested by $\mathcal{A}$ to the oracle $\mathsf{Sign}(s, \cdot)$ which returns $\sigma_i \xleftarrow{\$} \mathsf{Sign}(s, m_i)$ for all $i \in \{1, \ldots, \kappa\}$.

We also make use of the UC-signature functionality proposed in [38] that we denote with $\mathcal{F}_{\mathtt{SIGN}}$. We also use the the fact that a scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ that satisfies Def. 3 can be turned into a scheme that UC-realized the functionality $\mathcal{F}_{\mathtt{SIGN}}$ [38, Thm 2]. We assume familiarity with $\mathcal{F}_{\mathtt{SIGN}}$, and for the formal definition of this functionality we refer the reader to Fig. 2.4, Sec 2.4.1.4.

2.2 Privacy

In this section, we introduce definitions and notation for differential privacy (DP), distributional differential privacy (DDP). The notation will be used in the next section in the context of voting.

2.2.1 Differential privacy

For any set $\mathcal{U}$ and $n \in \mathbb{N}$, a *dataset* $\vec{X} \in \mathcal{U}^n$ of size n is a n-tuple where each entry is from $\mathcal{U}$. We call $\mathcal{U}$ a *universe*. We denote the ith entry of a dataset $\vec{X}$ as $\vec{X}_i$, and say that two datasets $\vec{X}, \vec{X'} \in \mathcal{U}^n$ are *neighbors*, if they differ by exactly one entry: there is exactly one i where $\vec{X}_i \neq \vec{X}'_i$. The range of a (randomized) function f, denoted $\mathbf{Range}(f)$ is the set of possible outputs of f.

Definition 4 (Differential privacy (DP) [58])**.** A function f is (ϵ, δ)-*differentially private* if for any pair of neighboring datasets $\vec{X}, \vec{X'}$, and any subset of outcomes $\mathcal{S} \subseteq \mathbf{Range}(f)$, the following inequality holds:

$$\Pr\left[f(\vec{X}) \in \mathcal{S}\right] \leq e^\epsilon \Pr\left[f(\vec{X'}) \in \mathcal{S}\right] + \delta. \tag{2.1}$$

DP has a useful property called *immunity to post-processing* [60]. Informally, this property says that if the function f is private, then so is any function g that only processes f's output.

Similar to DP, *distributional differential privacy*. In Ch. 3, we will prove this fact. We will then use it to make general statements about the privacy of voting rules, as corollary to the privacy of outputting vote tallies (we call this the *histogram*) in an election.

2.2.2 Distributional differential privacy

In context of distributional differential privacy, we consider dataset $\vec{X}$ as a random variable with some distribution π. We denote by $\vec{X}_{-i}$ as $\vec{X}$ but without the ith entry (also random variables). For any random variable, its support, denoted by $\mathbf{Supp}(\cdot)$ is the set of values the random variable can take with non-zero probability.

Definition 5 (Distributional differential privacy (DDP) [21])**.** A function f is $(\epsilon, \delta, \Delta)$-*distributional differentially private* if there is a function h[1] such that for all $D = (\pi, Z) \in \Delta$, $\vec{X} \sim \pi$, for all i, $(x, z) \in \mathbf{Supp}(\vec{X}_i, Z)$, and all sets $\mathcal{S} \subseteq \mathbf{Range}(f)$,

$$\Pr_{\vec{X}\sim\pi}(f(\vec{X}) \in \mathcal{S} | \vec{X}_i = x, Z = z) \leq e^{\epsilon} \Pr_{\vec{X}\sim\pi}(h(\vec{X}_{-i}) \in \mathcal{S} | \vec{X}_i = x, Z = z) + \delta$$

and

$$\Pr_{\vec{X}\sim\pi}(h(\vec{X}_{-i}) \in \mathcal{S} | \vec{X}_i = x, Z = z) \leq e^{\epsilon} \Pr_{\vec{X}\sim\pi}(f(\vec{X}) \in \mathcal{S} | \vec{X}_i = x, Z = z) + \delta$$

2.3 Voting

In this section, we define voting, using notation for datasets from our previous section on privacy. We introduce a class of voting rules called *generalized scoring rules* (GSR).

In the context of voting, the universe $\mathcal{U}$ models the set of possible ballots a voter can cast in an election. Given $n \in \mathbb{N}$, a dataset $\vec{X} \in \mathcal{U}^n$ of n votes is called a *preference profile*. We denote a vote in a profile $\vec{X}$ by $V \in \vec{X}$. If V is a vector, then V_i denotes the ith component of V. A voting rule for n voters is a function $f^v : \mathcal{U}^n \to \mathcal{O}$, where $\mathcal{O}$ denotes the set of outcomes of voting. We denote by $\mathcal{C} = \{c_1, \cdots, c_m\}$ a set of m candidates of an election. If $\mathcal{O} = \mathcal{C}$, we call the output of a voting rule f^v the *winner*.

An example of a voting rule is *plurality*. In plurality, $\mathcal{U} = \mathcal{C} = \mathcal{O}$; each voter votes for one favorite candidate, and the winner is the candidate with the most votes. Many voting rules have votes as linear orders, or rankings, over $\mathcal{C}$; that is, the set of all antisymmetric, transitive, and total binary relations over the set of candidates. We denote the set of all linear orders over $\mathcal{C}$ as $\mathcal{L}(\mathcal{C})$. An example of such a voting rule is *Borda*, where $\mathcal{U} = \mathcal{L}(\mathcal{C})$ and $\mathcal{O} = \mathcal{C}$. In Borda, each voter ranks the candidates according to his preferences, denoted by $c_{i_1} \succ c_{i_2} \succ \cdots \succ c_{i_m}$ where $a \succ b$ means a is preferred over b. The ith top-ranked candidate in each vote is awarded $m - i$ points, and the candidate with the most total points is the winner.

In elections, we are often also interested in a tally, or *histogram*, denoted by of the votes that have been cast.

[1] In [21] h is called the *simulator* in the sense that h "simulates" missing ith entry of $\vec{X}$, and following notation from a similar concept in security. However, to avoid confusion we simply refer to h as a function.

Definition 6 (Histogram)**.** Let $\mathcal{U} = \{x_1, \cdots, x_\ell\}$. For any $n \in \mathbb{N}$, the histogram function, denoted by $\mathbf{Hist} : \mathcal{U}^n \to \mathbb{N}$, takes as input a preference profile $\vec{X} \in \mathcal{U}^n$ and outputs a ℓ-dimensional integer vector whose ith component is $|\{j : \vec{X}_j = x_i\}|$.

2.3.1 Generalized scoring rules

Generalized scoring rules (GSR) [127] is a class of voting rules.

Definition 7 (Generalized scoring rules (GSR) [127])**.** A *generalized scoring rule* (GSR) is defined by a number $K \in \mathbb{N}$ and two functions: $f^{gsr} : \mathcal{L}(\mathcal{C}) \to \mathbb{R}^K$, and g^{gsr} which maps weak orders over the set $\{1, \cdots, K\}$ to $\mathcal{C}$. In GSR, a vote $V \in \mathcal{L}(\mathcal{C})$ is a ranking over the candidates, and we call $f^{gsr}(V)$ the *generalized score vector* of V. Given a profile $\vec{X}$, we call $f^{gsr}(\vec{X}) = \sum_i f^{gsr}(\vec{X}_i)$ the *score*. Then, the winner of a GSR is given by $g^{gsr}(\mathbf{Ord}(f^{gsr}(\vec{X})))$, where $\mathbf{Ord}$ outputs the weak order of the K components in $f^{gsr}(\vec{X})$.

From this definition, we see that all GSRs satisfy voter anonymity—that is, the identity of the voter does not matter. As such, it is sufficient to analyse GSRs in terms of the histogram of votes (rather than the preference profile). Most popular voting rules, e.g., Borda, Plurality, k-approval and ranked pairs, are GSRs. We give two examples of how to describe voting rules as GSRs.

Example 1 (Plurality as a GSR)**.** The simplest example of a GSR is plurality. Recall this is the voting rule where each voter chooses exactly one candidate, and the candidate with the most votes is the winner. Here, $K = m$, the number of candidates. Suppose V is a vote (linear order over candidates) where the top candidate is c_i. The function f^{gsr} would map V to a vector $f^{gsr}(V) = (0, \cdots, 0, 1, 0, \cdots, 0)$ where the only 1 is at the ith position of the vector $f^{gsr}(V)_i$. Then, for a profile $\vec{X}$, $f^{gsr}(\vec{X})$ is exactly the histogram counting the number of times each candidate is ranked at the top of a vote. Finally, the function g^{gsr} chooses the winner as the candidate with the most top-ranked votes, and ties can be broken lexicographically.

Example 2 (Borda as a GSR)**.** For Borda, we have $K = m$. Given a vote V, $f^{gsr}(V)$ is the vector where if the ith candidate c_i is the jth top-ranked candidate in V, then $f^{gsr}(V)_i = m - j$. That is, the candidate i receives $m - j$ points from vote X. Then, $f^{gsr}(\vec{X})$ records the total number of points each candidate receives from the profile $\vec{X}$. Finally, g^{gsr} chooses the winner as the candidate with the most points, and ties can be broken lexicographically.

A common rule that is not a GSR is Dodgson's rule, which does not satisfy homogeneity. A rule that does not satisfy voter anonymity (and thus not a GSR) is dictatorship, where only the vote of one voter counts.

2.4 Cryptographic security in the universal composition (UC) framework

One of the most utilized and accepted notions of security is universal composability (UC), which can be augmented with a global setup (GUC) that facilitates analyzing the security of protocols with global trusted setups such as a public key infrastructures (PKIs). The core of (GUC) security is the indistinguishability between the real and ideal worlds. In the real world, parties execute a protocol Π and communicate over a channel defined in the model. In the ideal world, parties access a functionality $\mathcal{F}$ which obtains inputs from them and returns to them the output directly. In both worlds, parties may have access to a global setup which can, for example, be a public key infrastructure (PKI). A protocol Π *securely-realizes* a functionality $\mathcal{F}$ if any adversary A in the real world can be emulated by a simulator S in the ideal world. That is, there does not exist any algorithm (which we call the environment Z) which can distinguish between the transcript of events in the two worlds. In addition, a protocol Π is *composable* if Π remains secure, even after replacing its calls to $\mathcal{F}$ (a functionality) with calls to a protocol Π' that securely realizes $\mathcal{F}'$. In this case, we call Π a $\mathcal{F}'$-hybrid protocol.

2.4.1 Security in the UC model

2.4.1.1 Multiparty computation (MPC)

Following [23] we consider MPC protocols where at each round ℓ, each party P_i broadcasts (see Fig. 2.2) a message $\mathtt{msg}_i^\ell$ to all the other parties simultaneously.

Definition 8 (MPC protocol)**.** Let n be a positive integer, m a polynomial in the security parameter κ, and $\mathcal{F}$ an n party-functionality. An m-round, n-party MPC protocol Π^{MPC} for $\mathcal{F}$ can be described as a tuple of deterministic polynomial-time algorithms $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$.

Next message: $\mathtt{msg}_i^\ell \leftarrow \mathsf{Next}_i(1^\kappa, x_i, \rho_i^\ell, \overline{\mathtt{msg}}^{<\ell})$ is the message broadcasted by party $p_i \in \mathcal{P}$ in round $\ell \in [m]$, on input $x_i \in \{0,1\}^\kappa$, on random tape $\rho_i^\ell \xleftarrow{\$} \{0,1\}^\kappa$, after receiving the messages $\overline{\mathtt{msg}}^{<\ell} = \{\mathtt{msg}_j^{\ell'}\}_{j\in[n],\ell'<\ell}$, where $\mathtt{msg}_j^{\ell'}$ is the message broadcasted by party p_j on round $\ell' \in [\ell-1]$. When $\overline{\mathtt{msg}}^{<m+1} = \{\mathtt{msg}_j^{\ell'}\}_{j\in[n],\ell'<m+1}$ then $y_i \leftarrow \mathsf{Next}_i(1^\kappa, x_i, \rho_i^{m+1}, \overline{\mathtt{msg}}^{<m+1})$ where y_i denotes the output of the party p_i.

2.4.1.2 Universal composability with global setup (GUC)

Below, we formally state the definition of the GUC framework from Canetti et al. [39]. Let $\mathcal{R}$ and $\bar{\mathcal{G}}$ be functionalities. Let Π be a $\mathcal{R}$-hybrid protocol, $\bar{\mathcal{G}}$ a setup, A an adversary, and Z an environment. The output of the environment Z after an execution of Π in the GUC $\bar{\mathcal{G}}$-hybrid model in presence of A is denoted as $\mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathrm{A},\mathrm{Z}}$. The output of Z in the ideal world where the simulator S interacts with an ideal functionality $\mathcal{F}$ and the setup $\bar{\mathcal{G}}$ is denoted as $\mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\Pi,\mathrm{S},\mathrm{Z}}$. These are both random variables (r.v.). We denote by $\approx$ to mean the two random variables are *computationally indistinguishable*. That is, the probability that any PPT (in κ) algorithm

outputs 1 when interacting with one r.v., and the probability that the algorithm outputs 1 after interacting with the other r.v., differ by at most a negl(κ) probability.

Definition 9 (UC with global setup (GUC))**.** Let $\bar{\mathcal{G}}$ be a global setup, and $\mathcal{R}$ be a resource. For an n-party efficient protocol Π (e.g., see Sec. 2.4.1.1) and functionality $\mathcal{F}$, we say that Π *GUC-realizes* $\mathcal{F}$ if $\forall \mathrm{A}\ \exists \mathrm{S}\ \forall \mathrm{Z}$

$$\mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathrm{A},\mathrm{Z}} \approx \mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\mathrm{S},\mathrm{Z}}$$

2.4.1.3 Security with identifiable (unanimous) abort

The notion of secure function evaluation (SFE) with *identifiable abort* allows the computation to fail (abort), but ensures that when this happens all the honest parties are informed about it, and they also agree on the index i of some corrupted party $p_i \in \mathcal{P}$ [87]. We denote the ideal functionality for the evaluation of the function f that captures the property of identifiable abort with $\mathcal{F}^f_{\mathsf{IDA}}$ (where IDA stands for identifiable abort). We also consider the notion of *unanimous abort*. This guarantees that either all or none of the honest parties abort. We denote the ideal functionality for the evaluation of the function f that captures the property of unanimous abort with $\mathcal{F}^f_{\mathsf{UNA}}$ (where UNA stands for unanimous abort). We refer to [49] for a more detailed discussion of the notions of security with non-unanimous (aka selective), unanimous, and identifiable abort, and their relation. In Sec. 2.4.1.4, we formally describe the functionalities for SFEs and identifiable/unanimous abort.

We will assume synchronous computation, i.e., our protocols proceed in rounds, where in each round: the uncorrupted parties generate their messages for the current round, as described in the protocol; then the messages addressed to the corrupted parties become known to the adversary; then the adversary generates the messages to be sent by the corrupted parties in this round; and finally, each uncorrupted party receives all the messages sent in this round. Although our treatment is in the (G)UC setting, to avoid overcomplicating the exposition, we will use the standard round-based language of [35, 113] to specify our protocol. Notwithstanding, such specifications can be directly translated to the synchronous UC model of Katz *et al.* [91] by assuming a clock functionality and bounded (zero) delay channels. We refer the interested reader to [91] for details.

2.4.1.4 Some useful functionalities

We provide formal definitions of several useful functionalities. In Fig. 2.1 and 2.2 we provide the SFE and broadcast functionalities proposed in [65]. In Fig 2.3 we provide the formal description of the for unanimous abort functionality $\mathcal{F}^{\mathsf{UNA}}$ (the description of the functionality extends to 3-party functionality provided in [117]). The formal description for the identifiable abort functionality $\mathcal{F}^{\mathsf{IDA}}$ is similar, except on abort, all honest parties output the ID of one corrupt party (this party

$\mathcal{F}_{\mathsf{SFE}}^{f}$ is as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^n \times R \to (\{0,1\}^*)^n$ and a set of parties $\mathcal{P}$. Initialize the variables $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ to a default value $\perp$.

- Upon receiving (Input, v) from some party $p_i \in \mathcal{P}$, set $x_i := v$ and send a message (Input, i) to the adversary.
- Upon receiving (Output) from some party $p_i \in \mathcal{P}$, do:
  1. If x_j has been set for all $j \in \mathcal{H}$, and $y_1, \ldots, y_n$ have not yet been set, then choose $r \xleftarrow{\$} R$ and set $(y_1, \ldots, y_n) := f(x_1, \ldots, x_n, r)$.
  2. Output y_i to p_i

Figure 2.1: The $\mathcal{F}_{\mathsf{SFE}}^{f}$ functionality of [65]

$\mathcal{B}$ is as follows, given a set of parties $\mathcal{P}$.

- Upon receiving x_i from party $p_i \in \mathcal{P}$, send (x_i, p_i) to every party in $\mathcal{P}$. (If $\mathcal{B}$ is considered a UC functionality, the output is given in a delayed manner, cf. [36])

Figure 2.2: The broadcast functionality $\mathcal{B}$ of [64, 85]

- Upon receiving (Input, v) from some party $p_i \in \mathcal{P}$, set $x_i := v$ and send a message (Input, i) to the adversary. If v is outside the domain of p_i consider $x_i = \texttt{ABORT}$.
- If there exists $i \in \{1, \ldots, n\}$ such that $x_i = \texttt{ABORT}$ the set $(y = \perp)$ else set $y = f(x_1, \ldots, x_n)$ and send y to the adversary.
- Upon receiving ok from the adversary, send y to the honest parties (if they query the functionality to get the output).
- Upon receiving ABORT from the adversary send $\perp$ to the honest parties (if they query the functionality to get the output)

Figure 2.3: The $\mathcal{F}^{\mathsf{UNA}}$.

may or may not be the corrupt party who caused the abort). In Fig. 2.4, we provide the basic UC signature functionality proposed in [38] modified to support strong unforgeability.

Key Generation.
Upon receiving a value $(\texttt{KEY\_GEN}, \text{sid})$ from some party $S \in \mathcal{P}$, verify that $\text{sid} = (S, \text{sid}')$ for some sid. If not, then ignore the request. Else, hand $(\texttt{KEY\_GEN}, \text{sid})$ to the $\mathcal{A}$. Upon receiving $(\texttt{VERIFICATION\_KEY}, \text{sid}, v)$ from the $\mathcal{A}$, output $(\texttt{VERIFICATION\_KEY}, \text{sid}, v)$ to S, and record the pair (S, v).
Signature. If $I = (\text{SIGN}, \text{sid}, m)$ is received from party S, verify that $\text{sid} = (S, \text{sid}')$ for some sid$'$. If not, then ignore the request, else send $(\text{SIGN}, \text{sid}, m)$ to $\mathcal{A}$. Upon receiving $I = (\texttt{SIGNATURE}, \text{sid}, m, \sigma)$ from $\mathcal{A}$, verify that no entry $(m, \sigma, v, 0)$ is stored. If it is, then output an error message to S and halt. Else, send $(\texttt{SIGNATURE}, \text{sid}, m, \sigma)$ to S, and store the entry $(m, \sigma, v, 1)$.
Verification
Upon receiving a value $(\texttt{VERIFY}, \text{sid}, m, \sigma, v')$ from some party p_i, hand $(\texttt{VERIFY}, \text{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\texttt{VERIFIED}, \text{sid}, m, \phi)$ from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)

2. Else, if $v' = v$, the signer is not corrupted, and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees strong unforgeability: If v' is the registered one, the signer is not corrupted, and a signature σ of m has never been generated, then the verification fails.)

3. Else, if there is an entry (m, σ, v', f') stored, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)

4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ)

Send $(\texttt{VERIFIED}, \text{sid}, m, f)$ to p_i.

Figure 2.4: The $\mathcal{F}_{\texttt{SIGN}}$ functionality with strong unforgeability of [38]

2.5 The Bitcoin backbone protocol

The abstraction of the Bitcoin protocol that is used in the cryptographic literature is known as the *Bitcoin backbone protocol* [17, 66, 115] which we denote by $\Pi^{\text{Ƀ}}$. In this abstraction, Bitcoin is modeled as a round-based protocol, where a number of participants (the miners) are connected via a multicast network (see Sec. A.2) with bounded delay of Δ rounds (unknown to the protocol). In every round, each party adopts the longest chain $\mathcal{C} = B_0 || \ldots || B_k$ of block B_i (connected by hash-pointers) it has received so far from the multicast network, where B_0 is the unique genesis block of the system. Each party tries to extend this longest chain an by additional block, via running the PoW-lottery: an extension of chain $\mathcal{C}$ by a new block B_{k+1} can only be valid, if

its hash [2] $H(B_{k+1})$ belongs to a dedicated small portion of the output domain of the function (typically, the hash must have a lot of leading zeros). In such analyses, the hash function is modeled using a random-oracle functionality $\mathcal{F}_{\mathrm{RO}}$ that returns uniform values upon each query. Therefore, when extending the chain, each party makes a certain number of *mining queries* per round (that is, RO-queries with candidate blocks B_{k+1} containing a random nonce to obtain the hash) and we call a mining query *successful*, if the output is below the threshold. In the setting with fixed PoW difficulty, we can assign a success probability p to each such mining query. Finally, if a miner is successful, it will send the new chain over the multicast network to all other miners.

Cryptographic security. The main security guarantee[3] proven for the Bitcoin protocol is eventual *consistency*: every block that is deep enough can be considered immutable and only the most recent, cutOff number of blocks might be transient. This cutOff-consistency (where the cutoff parameter is often left implicit if clear from context) guarantee states that at any point in time, the prefix of $\mathcal{C}$ consisting of $|\mathcal{C}| - \text{cutOff}$ blocks is agreed-upon by all honest miners:

Definition 10 (Consistency)**.** Let $\mathcal{C}_1 \preccurlyeq \mathcal{C}_2$ denote the prefix-of relation, then the consistency guarantee (with parameter cutOff) states that at any two points in time $a \leq b$ in an execution, where party P at round a holds chain $\mathcal{C}_1$ and party P' at round b holds chain $\mathcal{C}_2$, we have that $\mathcal{C}_1|_{\mathsf{cutOff}} \preccurlyeq \mathcal{C}_2$, where the notation $\mathcal{C}|_k$ denotes the prefix of $\mathcal{C}$ obtained by removing the most recent k blocks (and if k exceeds the length of $\mathcal{C}$, it is defined to correspond to the genesis block).

In the cryptographic setting (without incentives), such a guarantee only holds if we restrict the adversary to have a minority of mining power. That is, given $n_a^{(r)}$ and $n_h^{(r)}$ denote the numbers of adversarial and honest mining queries in round r, respectively, then the protocol $\Pi^{\text{₿}}$ is secure if in any round r the inequality

$$n_a^{(r)} < \theta_{pow} \cdot n_h^{(r)}$$

holds, with

$$\theta_{pow} := (1-p)^{(2\Delta+1)\mathsf{T}_{\mathrm{ub}}}$$

being the well-established security threshold for Bitcoin (often stated in its linear approximation $1-2(\Delta+1)p\mathsf{T}_{\mathrm{ub}}$) [17,66,115], where the quantity T_{ub} denotes the upper bound on the number of mining queries per round. Throughout this thesis, we work in the so-called *flat model* of Bitcoin for notational simplicity [15, 66], where each miner gets one mining query per round (and the adversary's power is the number of corrupted miners). We note that sometimes it is convenient to assume a lower bound $\mathtt{T}_{\mathrm{lb}}$ on the number of mining queries (a.k.a. participation) per round, in particular when arguing about the guaranteed growth of the blockchain over time in combination with the security threshold. Finally, we point out that even if there are no adversarial players, an upper bound T_{ub} on the number of queries is necessary for security in the fixed difficulty setting, when aiming for a common prefix guarantee for some target parameter cutOff. As the failure probability of Bitcoin becomes negligible as a function of cutOff (more precisely, the relevant

[2] The hash function is idealized as the UC random-oracle (RO) functionality; see Sec. 2.5.1.1.

[3] While other security guarantees exist, such as *chain quality*, the thesis will focus (specifically Ch. 5) on consistency.

factor is of the order $2^{-\Omega(\mathsf{cutOff})}$), we often treat it as a (of course polynomial-bounded) function $\mathsf{cutOff}(\kappa)$ of a security parameter κ, and (in symbolic notation) $\mathsf{cutOff} = \omega(\log(\kappa))$ is at least required to obtain a negligible probability of a failure.

Bitcoin backbone and UC. The above Bitcoin backbone protocol $\Pi^{\text{฿}}$ is seen as a UC protocol as in [17], where it is proven to UC-realize a strong transaction ledger functionality $\mathcal{G}_{\text{LEDGER}}$ (see Sec. 2.5.1.3) under the honest majority assumption. We give here just the explanation of how the ideal consistency guarantee looks like: the functionality $\mathcal{G}_{\text{LEDGER}}$ ensures that at any point in time, there is only one unique ledger state (sequences of transactions packed in blocks), where the state is append-only (that is, whatever appears as a block in the state is immutable). Furthermore, different honest parties see different prefixes of this state, with the guarantee that these *views* are increasing and within a window of `windowSize` (a ledger functionality parameter) blocks from the tip of the state. Note that the cut-off parameter of Bitcoin corresponds exactly to the size of that window in the realized ledger $\mathcal{G}_{\text{LEDGER}}$. More precisely, whenever Bitcoin satisfies Definition 10, then the above mentioned correspondence holds and the ledger state is a single chain of blocks [17].

In UC, the protocol $\Pi^{\text{฿}}$ assumes a couple of hybrid functionalities. First, the round-based structure is achieved using UC-synchronous tools (assuming a clock functionality), a network, and a random oracle. In particular, in [17, 68] assumptions like honest majority and maximum queries per round T_{ub} (as well as minimum queries per round T_{lb}) can be captured by functionality wrappers, which restrict access to the RO such as disallowing number of accesses per (corrupt) party per round and total number of RO accesses per round. One extremely helpful aspect of UC in the context of RPD is the compatibility with the composition theorem [65]. This can be leveraged as follows. The Bitcoin backbone $\Pi^{\text{฿}}$ admits a modular structure that isolates the lottery aspect as a submodule of the system. Technically, the proofs in [17, 115] show that whenever the PoW-lottery UC-realizes the *state exchange* functionality $\mathcal{F}_{\text{STX}}$ (formally described in Sec. 2.5.1.2; in [115] the related concept is called $\mathcal{F}_{\text{tree}}$), the Nakamoto-style longest chain rule protocol (under the above honest-majority security threshold) realizes the ledger functionality. This intermediate step is important due to two things: first, it models an idealized mining process where each mining query is an independent Bernoulli trial with success probability p (and hence abstracts away those real-life negligible probability events that would destroy independence), and second it abstracts away the low-level details of the chain structure (where e.g., "hash collisions" could cause disruptions). It is proven in [17] that the proof-of-work layer of Bitcoin (in the random oracle model) UC-realizes $\mathcal{F}_{\text{STX}}$. Moreover, since it only abstracts the lottery part of the system, this realization does not depend on any security threshold. We can therefore leverage composition when analyzing the utilities of Bitcoin even when honest majority may not hold, and work with the idealized lottery directly.

2.5.1 Formal description of Bitcoin in UC

The UC Bitcoin protocol uses several idealized functionalities [17]: the clock—which keeps track of the current round and multicast (which allows parties to send messages to multiple other

parties),. Since the specifics of these functionalities are not important to the results in the thesis, for completeness we include them in App. A. We also include random oracle, the ideal lottery, the ledger functionality, as well as a weaker (in particular non-consistent) ledger functionality which is achieved when honest majority does not hold, pointing out relevant details and differences between the two ledgers.

2.5.1.1 Random oracle

In Fig. 2.5 we formally describe the random oracle functionality, which parties query to mine blocks in the UC. In order to model the limit on the total mining power in the system, we also consider a *wrapped* RO, which is the RO, but in addition restricts the number of queries per party per round.

The functionality $\mathcal{F}_{\mathrm{RO}}$ is parametrized by a security parameter κ. It maintains a set of registered parties/miners $\mathcal{P}$ (initially set to $\emptyset$) and a (dynamically updatable) function table H (initially $H = \emptyset$). For simplicity we write $H = \perp$ to denote the fact that no pair of the form $(x, \cdot)$ is in H.

Upon receiving $(\mathrm{EVAL}, \mathsf{sid}, x)$ from some party $p \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of a corrupted p), do the following:

1. If $H[x] = \perp$ sample a value y uniformly at random from $\{0,1\}^{\kappa}$, set $H[x] \leftarrow y$ and add $(x, H[x])$ to H.
2. Return $(\mathrm{EVAL}, \mathsf{sid}, x, H[x])$ to the requestor.

Figure 2.5: The functionality $\mathcal{F}_{\mathrm{RO}}$

2.5.1.2 Lottery functionality $\mathcal{F}_{\mathsf{StX}}$

The lottery functionality $\mathcal{F}_{\mathrm{StX}}^{\Delta, p_H, p_A}$ (Fig. 2.6, as defined in [17]) abstracts the process of choosing a (random) party to generate the next block in the ledger. Here, Δ is the network delay, p_H and p_A are the probabilities for successfully creating a block for each mining attempt—in the flat model, as is in our setting, $p_H = p_A = p$ (which is a protocol parameter that relates to the *difficulty* of the blockchain). Roughly, the state exchange functionality manages a tree structure containing successfully-mined chains (thus eliminating issues with hash collisions when using hash pointers in the chain), and abstracts mining with random oracle (RO) queries as coin tosses with probability of success p. It also manages the multicasting of successfully-mined chains, and allows the adversary to influence the network via network delay.

Note that in this hybrid world, defining restrictions on the total number of mining queries in the system is straightforward by introducing a function wrapper $\mathcal{W}_{\mathrm{T_{lb}}}^{\mathrm{T_{ub}}}$. The wrapper can limit

mining queries (i.e. calls to the RO or queries to SUBMIT-NEW) if T_{ub} is reached, and can ensure that the system only advances to the next round if at least T_{lb} queries have been submitted. We refer to [17, 68] for more details on wrappers.

2.5.1.3 Ledger functionality

For completeness, we describe the ledger functionality $\mathcal{G}_{\mathrm{LEDGER}}$ from [17] in App. A. $\mathcal{G}_{\mathrm{LEDGER}}$ is parametrized by four algorithms, Validate, `ExtendPolicy`, Blockify, and $\texttt{predict-time}_{BC}$. Their details are not relevant to this thesis, so we describe their function informally: Validate determines whether a (new) block is valid according to the current ledger state (e.g. no double-spending transactions). `ExtendPolicy` ensures that the ledger state, with the proposed new block, would still satisfy desirable properties like *chain quality*. Blockify collects transactions into a block, and $\texttt{predict-time}_{BC}$ is required to ensure the ledger functionality advances a round when the protocol does. The functionality also has two parameters: `windowSize` and $\texttt{Delay} \in \mathbb{N}$. The parameter `windowSize` is the consistency parameter, and is generally the same as the cut-off parameter cutOff of the Bitcoin backbone protocol. The parameter $\texttt{Delay} \in \mathbb{N}$ depends on the network delay Δ, and is set as $\texttt{Delay} := 4\Delta$ in the main theorem of [17].

2.5.1.4 Weak ledger functionality

It is useful to define a weaker ledger functionality which is achieved when honest majority does not hold. Below, we informally describe this weaker Bitcoin ledger functionality $\mathcal{G}^{\bitcoin}_{\mathrm{WEAK\text{-}LEDGER}}$ from [15], which does not enforce many security properties, and include the formal description in App. A.

Most importantly, this weaker ledger allows the simulator to break *consistency* via a FORK command. Looking ahead, this functionality will be useful in our rational analyses of the blockchain under 51% attack, as we will analyze scenarios when this FORK command is necessary (or not) to simulate a rational adversary. This in turn will tell us whether consistency can be achieved under said rational adversary.

Similar to the (strong) ledger functionality $\mathcal{G}_{\mathrm{LEDGER}}$, the weak ledger functionality $\mathcal{G}^{\bitcoin}_{\mathrm{WEAK\text{-}LEDGER}}$ is parametrized by four algorithms, Validate, weakExtendPolicy, Blockify, and `predict-time`, along with two parameters: $\texttt{windowSize}, \texttt{Delay} \in \mathbb{N}$. The main difference is that `ExtendPolicy` is replaced with its weaker version weakExtendPolicy. Below, we describe these defining features of the functionality:

State tree: Instead of storing a single ledger state state, $\mathcal{G}^{\bitcoin}_{\mathrm{WEAK\text{-}LEDGER}}$ stores a tree `state-tree` of state blocks where for each node the direct path from the root defines a ledger state. The functionality maintains for each registered party $p_i \in \mathcal{P}$ a pointer $\mathtt{pt}_i$ to a node in the tree which defines the current-state view of p_i which the adversary can set. The pointer of a honest party can only be set to a node which has a at least the distance to the root of the current pointer node.

The functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ is parametrized with a set of parties $\mathcal{P}$, keeping track of registered (resp. deregistered) parties. Each $p \in \mathcal{P}$ manages a tree $\mathcal{T}_p$ which initially contains the genesis state, and each rooted path corresponds to a valid state the party has received. Finally, it manages a buffer $\vec{M}$ containing successfully submitted states not yet been delivered to (some) parties in $\mathcal{P}$, and buffer $\mathbf{N}_{\text{net}}$ of adverbially injected chunk messages (that might not correspond to valid states).

Submit/receive new states:

- Upon receiving $(\text{SUBMIT-NEW}, \mathsf{sid}, \vec{\mathtt{st}}, \mathtt{st})$ from some participant $p_s \in \mathcal{P}$, if isvalidstate$(\vec{\mathtt{st}}||\mathtt{st}) = 1$ and $\vec{\mathtt{st}} \in \mathcal{T}_p$ do the following:
  1. Sample B from Bernoulli-Distribution with parameter p_H (or p_A for dishonest p_s).
  2. If $B = 1$, set $\vec{\mathtt{st}}_{new} \leftarrow \vec{\mathtt{st}}||\mathtt{st}$ and add $\vec{\mathtt{st}}_{new}$ to $\mathcal{T}_{p_s}$. Else set $\vec{\mathtt{st}}_{new} \leftarrow \vec{\mathtt{st}}$.
  3. Output $(\text{SUCCESS}, \mathsf{sid}, B)$ to p_s.
  4. On response $(\text{CONTINUE}, \mathsf{sid})$ where $\mathcal{P} = \{p_1, \ldots, p_n\}$ choose n new unique message-IDs $\mathsf{mid}_1, \ldots, \mathsf{mid}_n$, initialize n new variables $D_{\mathsf{mid}_1} := D_{\mathsf{mid}_1}^{MAX} := \ldots := D_{\mathsf{mid}_n} := D_{\mathsf{mid}_n}^{MAX} := 1$ set $\vec{M} := \vec{M}||(\vec{\mathtt{st}}_{new}, \mathsf{mid}_1, D_{\mathsf{mid}_1}, p_1)|| \ldots ||(\vec{\mathtt{st}}_{new}, \mathsf{mid}_n, D_{\mathsf{mid}_n}, p_n)$, and send $(\text{SUBMIT-NEW}, \mathsf{sid}, \vec{\mathtt{st}}_{new}, p_s, (p_1, \mathsf{mid}_1), \ldots, (p_n, \mathsf{mid}_n))$ to the adversary.
- Upon receiving $(\text{FETCH-NEW}, \mathsf{sid})$ from a party $p \in \mathcal{P}$ or $\mathcal{A}$ (on behalf of p), do the following:
  1. For all tuples $(\vec{\mathtt{st}}, \mathsf{mid}, D_{\mathsf{mid}}, p) \in \vec{M}, \mathbf{N}_{\text{net}}$ update value $D_{\mathsf{mid}} := D_{\mathsf{mid}} - 1$.
  2. Let $\vec{M}_0^p$ denote the subvector of $\vec{M}$ including all tuples of the form $(\vec{\mathtt{st}}, \mathsf{mid}, D_{\mathsf{mid}}, p)$ where $D_{\mathsf{mid}} = 0$ (in the same order as they appear in $\vec{M}$). For each tuple $(\vec{\mathtt{st}}, \mathsf{mid}, D_{\mathsf{mid}}, p) \in \vec{M}_0^p$ add $\vec{\mathtt{st}}$ to $\mathcal{T}_p$. Delete all entries in $\vec{M}_0^p$ from $\vec{M}$ and send $\vec{M}_0^p$ to p. If p is corrupted, provide additionally $\mathbf{N}_{\text{net}}$ to the adversary.
- Upon receiving $(\text{SEND}, \mathsf{sid}, \vec{\mathtt{st}}, p')$ from $\mathcal{A}$ on behalf some *corrupted* $p \in \mathcal{P}$, if $p' \in \mathcal{P}$ and $\vec{\mathtt{st}} \in \mathcal{T}_p$, choose a new unique message-ID mid, initialize $D := 1$, add $(\vec{\mathtt{st}}, \mathsf{mid}, D_{\mathsf{mid}}, p')$ to $\vec{M}$, and return $(\text{SEND}, \mathsf{sid}, \vec{\mathtt{st}}, p', \mathsf{mid})$ to $\mathcal{A}$. If $\vec{\mathtt{st}} \notin \mathcal{T}_p$, then conduct the same steps except that $(\vec{\mathtt{st}}, \mathsf{mid}, D_{\mathsf{mid}}, p')$ is added to $\mathbf{N}_{\text{net}}$.

Figure 2.6: The functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ (Part 1)

(Continued from description of $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$, Part 1)

Further adversarial influence on the network:

- Upon receiving $(\text{SWAP}, \text{sid}, \text{mid}, \text{mid}')$ from $\mathcal{A}$, if mid and mid$'$ are message-IDs registered in the current message buffers, swap the corresponding tuples in the buffer. Return $(\text{SWAP}, \text{sid})$ to $\mathcal{A}$.
- Upon receiving $(\text{DELAY}, \text{sid}, T, \text{mid})$ from $\mathcal{A}$, if T is a valid delay, mid is a message-ID for a tuple $(\vec{\text{st}}, \text{mid}, D_{\text{mid}}, p)$ in a message buffer and $D_{\text{mid}}^{MAX} + T \leq \Delta$, set $D_{\text{mid}} := D_{\text{mid}} + T$ and set $D_{\text{mid}}^{MAX} := D_{\text{mid}}^{MAX} + T$.

Figure 2.7: The functionality $\mathcal{F}_{\text{StX}}^{\Delta,p_H,p_A}$ (Part 2)

Adding transactions: Submitted transactions are simply collected in buffer without any additional check. Transactions in buffer which are added to `state-tree` are not removed as they could be reused at an other branch of `state-tree`.

Adding blocks and forking: The command NEXT-BLOCK which allows the adversary to propose the next block takes additionally a leaf of `state-tree` as input which defines where the next block will be added. By default the next block is added to the longest branch of `state-tree`. To add the next block to an intermediate node of `state-tree` the adversary may use the command FORK which otherwise provides the same functionality as NEXT-BLOCK.

Extend-policy: The weak extend-policy weakExtendPolicy that is invoked when the ledger extends one of its branches checks a few simple validity validity conditions. It takes a state-tree `state-tree` and pointer `pt` as input. It first computes a default block $\vec{N}_{\text{df}}$ which can be appended at the longest branch of `state-tree` without rendering the state invalid. Then it checks if the proposed blocks (by the adversary) $\vec{N}$ can be safely appended at the node `pt` without violating the validity of the chain. If this is the case it returns $(\vec{N}, \texttt{pt})$. Otherwise it returns the $\vec{N}_{\text{df}}$ and a pointer to the leaf of the longest branch in `state-tree`. In contrast to strong ledger in [17], the weak extend-policy does not check if the adversary inserts too many or too few blocks, does not give guarantees whether old transactions will be included, and does not enforce a fraction of honest blocks.

The ledger further obtains time-stamps from a clock functionality (Fig. A.1).

2.6 Cryptography and game theory

2.6.1 Implementing games without collusion

Collusion-freeness (CF) was introduced by Lepinski et al. [100] to capture the requirement of non-collusion when implementing a game of poker in a cryptographic protocol. Lepinski et al. identified the issue of using the standard cryptographic security definition: malicious parties are always assumed to be able to coordinate by having a direct channel to share information, or *collude*, with each other (otherwise known as a *monolithic* adversary). In other words, a secure protocol is not necessarily a collusion-free protocol.

Here, we will focus on the composable version of CF, called *collusion-preservation* (CP), which extends the idea of CF with Global UC (GUC), which we introduced in the previous section. Similar to the CF, CP requires the lack of collusion among corrupt parties, by replacing the monolithic adversary in (G)UC with a set of independent malicious parties who must communicate only with the available channels described in the protocol or functionality.

2.6.1.1 Collusion-preservation

Collusion-preserving computation (CP), proposed in [9], differs from (G)UC in the following manner. For n the number of parties, the malicious attacker is described as a set of adversarial strategies (which we call for convenience *adversaries*), instead of one monolithic adversary/simulator (as in (G)UC) which controls all corrupt parties. In more detail, we consider a set of independent PPT adversaries and simulators, which requires the following:

- **Split adversaries/simulators:** Instead of a monolithic adversary/simulator we consider a set of n polytime adversaries $\mathrm{A}_{[n]} = \{\mathrm{A}_i, i \in [n]\}$, where A_i corresponds to the adversary associated with the player i–and can corrupt at most this party. We also ask that for each $\mathrm{A}_i \in \mathrm{A}_{[n]}$ there exists an (independent) simulator S, who only has access to A_i.

- **Corrupted-set independence:** We also require that the simulators do not depend on each other. In other words the code of S_i is the same for any set of adversaries $\mathrm{A}_{[n]}$ and $\mathcal{B}_{[n]}$ as long as $\mathrm{A}_i = \mathcal{B}_i$.

Similar to the GUC framework (but in contrast to plain UC) we distinguish between two types of functionalities: resources, denoted with capital calligraphic font as in $\mathcal{R}$, and shared functionalities, denoted with an additional over-line as in $\bar{\mathcal{G}}$. Formally, a resource $\mathcal{R}$ maintains state only with respect to a single instance of a protocol, while a shared functionality $\bar{\mathcal{G}}$ can maintain state across protocol instances. For example, concurrent executions can maintain shared state via e.g., a global CRS or global PKI as long as they are modeled as shared functionalities. However, although concurrent instances of a protocol Π may use the same resource $\mathcal{R}$, the behavior of $\mathcal{R}$ in one execution of Π must be independent of all other executions of Π (and more generally of all other concurrent protocols instantiated by the environment). For clarity, in the remainder of this thesis we will usually refer to shared functionalities simply as *setup*, and protocols that share state across executions only through some setup $\bar{\mathcal{G}}$ as $\bar{\mathcal{G}}$*-subroutine*

respecting. We denote by $\text{CP-EXEC}^{\mathcal{R}}_{\Pi,\mathrm{A},\mathrm{Z}}$ the output of the environment Z in the execution of Π with adversaries $\mathrm{A} := \mathrm{A}_{[n]}$ in the $\mathcal{R}$-hybrid model. We say that a protocol Π is $\mathcal{R}$-*exclusive* if it makes use of no resources or shared functionality other than $\mathcal{R}$. We note that unlike (G)UC, CP limits parties to communicate with at most one single instance of the resource. Intuitively, if $\mathcal{F}$ is a one-bit channel, then the simulator only using one instance of $\mathcal{F}$ has a completely different meaning in terms of collusion-preservation to the simulator using unlimited calls to $\mathcal{F}$. As in the definition of UC security, we denote two random variables as computationally indistinguishable by the notation $\approx$.

Definition 11 (Collusion-preservation [9])**.** Let $\bar{\mathcal{G}}$ be a setup, $\mathcal{R}$ and $\mathcal{F}$ be n-party resources, Π be a $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol and ϕ be a $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol (both with n parties). Then we say that Π collusion-preservingly (CP) emulates ϕ in the $\{\bar{\mathcal{G}}, \mathcal{F}\}$-hybrid world, if there exists a collection of efficiently computable transformations $\mathrm{Sim} = \{\mathrm{Sim}_1, \ldots, \mathrm{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathrm{A} = \{\mathrm{A}_1, \ldots, \mathrm{A}_n\}$, and every PPT environment Z the following holds: $\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathrm{A},\mathrm{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathrm{Sim}(\mathrm{A}),\mathrm{Z}}$

Following [9], we distinguish between the notion of *emulation* and its special case *realization*. For a functionality $\mathcal{F}$ we denote by $\mathcal{D}^{\mathcal{F}}$ the ith dummy $\mathcal{F}$-hybrid protocol which simply acts as a transparent conduit between the ith honest and adversarial interfaces of $\mathcal{F}$ and the environment Z. Specifically, $\mathcal{D}^{\mathcal{F}}$ forwards all messages it receives from Z to the $\mathcal{F}$ (where the choice of adversarial or honest interface is specified by Z) and vice-versa. If for functionality $\mathcal{F}$, an $\mathcal{R}$-hybrid protocol Π CP-emulates $\mathcal{D}^{\mathcal{F}}_i$ then we say that Π *realizes* $\mathcal{F}$ (in the $\mathcal{R}$-hybrid world).

2.6.2 Rational protocol design (RPD):

Security against incentive-driven attackers

In Chapters 4 and 5 we will adapt the rational protocol design framework (RPD) [15, 65] to study collusion-preserving games and blockchain incentives, respectively. The goal of RPD is to model security of protocols against incentive-driven attackers, especially when the desired security properties are impossible against general (non-rational) attackers. RPD is compatible with (G)UC security, and as we will show in Ch. 4 Sec. 4.3, RPD can also be modified to capture collusion-preservation (CP) against rational attackers. In Sec. 2.6.3, we will detail how the RPD framework was used in [15] to analyze the Bitcoin backbone protocol.

In RPD, a protocol designer $\mathtt{D}$ engages in an *attack game* $\bar{\mathcal{G}}$ with an attacker $\mathtt{A}$. The designer first chooses a (n-party) protocol $\Pi \in \mathtt{ITM}^n$. Then, based on Π, the attacker decides on an adversarial strategy A to attack Π. Each pair $(\Pi, \mathrm{A} = \mathtt{A}(\Pi))$ (called a *strategy profile* in line with game theory) induces a utility for the designer and the attacker. Consistent with [65], we consider an attack game $\bar{\mathcal{G}}$, where $\mathcal{M}$ is the attack model $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}})$, a vector of parameters of the game. $\mathcal{F}$ is the functionality which the designer would like to achieve, and $\langle\mathcal{F}\rangle$ is a relaxed version of $\mathcal{F}$ in the sense that it includes extra commands that break certain security properties of $\mathcal{F}$. The value function $v_{\mathtt{A}}$ allows us to define utilities of the attacker. It assigns payoffs when certain events occur in the ideal world, such as when the simulator uses the extra commands in $\langle\mathcal{F}\rangle$ to help him complete a successful simulation.

The goal when analyzing in RPD is ensure that it is not in the attacker's best interest to force the simulator to use weaknesses of $\langle\mathcal{F}\rangle$. This is captured by the notion of *attack-payoff security*. A protocol $\langle\mathcal{F}\rangle$ is *attack-payoff secure* if no adversarial strategy (which we refer to as the *adversary* A) attacking Π can achieve more utility for the attacker than a strategy attacking a "dummy" protocol that just uses the functionality $\mathcal{F}$ (i.e. without weaknesses of $\langle\mathcal{F}\rangle$). In other words, Π is "as secure as" the dummy (trivially-secure) protocol in this model. Following [15], we can also augment $\mathcal{M}$ with the designer's value function $v_{\mathtt{D}}$, and consider whether the protocol is *incentive compatible* for the designer.

2.6.2.1 Utility of the attacker A

The utility of the attacker $u_{\mathtt{A}}$ is a function mapping protocols and sets of adversaries, i.e. the strategy profile $(\Pi, \mathrm{A}) = (\Pi, \mathtt{A}(\mathrm{A}))$, to a real number. More formally: First, we have a value function $v_{\mathtt{A}}$, defined in the attack model, which maps the views of the simulators and environment in the ideal world to a real value. Looking ahead, we define the real payoff of a particular A attacking the protocol, as the minimum payoff over all simulators that can emulate A. Finally, $u_{\mathtt{A}}(\Pi, \mathrm{A})$ is the real payoff of A, maximized over all possible environments Z.

Ideal payoff of the simulator In more detail, we define the real-valued random variable ensemble $\{v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathrm{S},\mathrm{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ (or $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathrm{S},\mathrm{Z}}$ for short) as the random variable ensemble resulting from applying value function $v_{\mathtt{A}}$ to the view of the environment Z and simulator S in the ideal execution. The ideal expected payoff of a particular simulator S with respect to Z is defined as the expected value: $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathrm{S},\mathrm{Z}) = E(v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathrm{S},\mathrm{Z}})$.

Real payoff of the adversary Recall that given a setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$, a protocol Π realizes some functionality $\langle\mathcal{F}\rangle$ if, for all adversaries A, there exists a simulator S that emulates A. That is, the environment Z cannot distinguish between the real world with A and resource $\mathcal{R}$, and ideal world with S and $\langle\mathcal{F}\rangle$.

Denote $\mathcal{S}_{\mathrm{A}}$ as the class of simulators S that can emulate the adversary A. In other words, for setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$,

$$\mathcal{S}_{\mathrm{A}} = \left\{\mathrm{S} \in \mathtt{ITM} | \forall \mathrm{Z}\colon \mathrm{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi,\mathrm{A},\mathrm{Z}} \approx \mathrm{EXEC}^{\bar{\mathcal{G}},\langle\mathcal{F}\rangle}_{\Pi,\mathrm{S},\mathrm{Z}}\right\}.$$

The expected payoff of a set of adversaries and environment (A,Z) is then defined as $U_{\mathtt{A}}^{\Pi,\langle\mathcal{F}\rangle}(\mathrm{A},\mathrm{Z}) = \inf_{\mathrm{S}\in C_{\mathrm{A}}}\{U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathrm{S},\mathrm{Z})\}$. The attacker's utility is then maximized over all environments Z, i.e., $u_{\mathtt{A}}(\Pi,\mathrm{A}) := \hat{U}_{\mathtt{A}}^{\Pi,\langle\mathcal{F}\rangle}(\mathrm{A}) = \sup_{\mathrm{Z}\in\mathtt{ITM}}\{U_{\mathtt{A}}^{\Pi,\langle\mathcal{F}\rangle}(\mathrm{A},\mathrm{Z})\}$.

2.6.2.2 Attack-payoff security in RPD

The work of [65] introduces the following notion of security against incentive-driven adversaries: No matter the utility achieved by an adversary $\mathcal{A}$ running the protocol Π in the real world, there exists an adversary $\mathcal{A}'$ running the dummy protocol with access to the ideal functionality $\mathcal{F}$ that achieves the same or better utility. In other words, even the best adversary attacking Π, cannot

achieve better utility than one who does not invoke any of the "bad events" in $\langle\mathcal{F}\rangle$. Note that here $\mathcal{F}$ can be any strengthening of its weaker version.

Definition 12 (Attack payoff security [65])**.** Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$ be an attack model inducing utility $u_{\mathtt{A}}$, and let $\Phi^{\mathcal{F}}$ be the dummy $\mathcal{F}$-hybrid protocol. A protocol Π is *attack-payoff secure* for $\mathcal{M}$ if for all $\mathcal{A}$, there is an $\mathcal{A}'$ such that $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\Phi^{\mathcal{F}}, \mathcal{A}') + \mathsf{negl}(\kappa)$

2.6.2.3 Security for the protocol designer D

On the side of the protocol designer, the definition that captures his willingness to follow the prescribed protocol Π is called *incentive compatibility* (IC) [15]. To define IC, we first define what is a ϵ-*subgame-perfect equilibrium* [65]. Informally, a strategy profile is an ϵ-subgame-perfect equilibrium if no deviation could improve utilities by more than ϵ. Intuitively, a protocol Π is incentive compatible when even the designer is incentivized to stick with it.

Definition 13 (ϵ-subgame-perfect equilibrium [65])**.** Let G be an attack game. A strategy profile $(\Pi, \mathtt{A}(\Pi))$ is an ϵ-*subgame perfect equilibrium* in G if the following conditions hold: (1) for any $\Pi' \in \mathtt{ITM}^n$, $u_{\mathtt{D}}(\Pi', \mathtt{A}(\Pi')) \leq u_{\mathtt{D}}(\Pi, \mathtt{A}(\Pi)) + \epsilon$, and (2) for any $\mathtt{A}' \in \mathtt{ITM}$, $u_{\mathtt{A}}(\Pi, \mathtt{A}'(\Pi)) \leq u_{\mathtt{A}}(\Pi, \mathtt{A}(\Pi)) + \epsilon$.

Definition 14 (Incentive Compatibility (IC))**.** Let Π be a $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol and $\vec{\Pi}$ be a set of polynomial-time $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocols. We say that Π is $\vec{\Pi}$-*IC* in the attack model $\mathcal{M}$ iff for some $\mathtt{A} \in \mathtt{ITM}$, $(\Pi, \mathtt{A}(\Pi))$ is a $\mathsf{negl}(\kappa)$-subgame perfect equilibrium on the restricted attack game where the set of deviations of the designer is $\vec{\Pi}$.

2.6.3 Rational treatment of the Bitcoin backbone protocol in RPD

The work of [15] analyzes the the Bitcoin protocol $\Pi^{\mathbb{B}}$ in RPD. We include some details of this analysis for completeness, and as an example application of the RPD framework. The Bitcoin backbone protocol $\Pi^{\mathbb{B}}$, detailed in Sec. 2.5, was shown in [15] to only implement a weaker ledger functionality $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ when the honest majority assumption is lifted. We recall that the weak ledger is derived from the stronger version [17] by introducing a few weaknesses. For example, it allows the adversary to fork the ledger state and hence allows it to break consistency (this event corresponds to a deep reorganization of the blockchain in the real world). This is allowed by the FORK command in $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$. Given the views of the simulator and environment in an ideal world execution, the value functions $v_{\mathtt{A}}$ and $v_{\mathtt{D}}$ assign payoffs to the attacker and designer respectively, when certain events happen in the views, such as when the simulator forks the blockchain via $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$. Looking ahead, in Ch. 5 we will also analyze $\Pi^{\mathbb{B}}$ in RPD, to find conditions under which a simulator would not need to invoke the consistency-breaking command of $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ in order to simulate rational attackers (i.e., attacks that break consistency are disincentivized). For example, if under a class of utilities, no rational attacker invokes the FORK command, then we essentially obtain a stronger ledger (i.e., the same except that this command is absent and hence the ledger state remains a unique chain) against attackers incentivized by this class of utilities.

2.6.3.1 Utility of the Attacker from [15]

We detail the attacker's utility in [15], which in the RPD framework captures the expected payoff of a particular adversarial strategy $\mathcal{A}$ in a given protocol Π (in this case $\Pi = \Pi^{\hat{\mathbb{B}}}$). This payoff is calculated based on different *events* that occur in the real execution and the corresponding ideal experiment where a black-box simulator is attempting to simulate this adversarial strategy.

Specifically, the work of [15] considers the following events:

1. Event $W^{\mathtt{A}}_{q,r}$, for each pair $(q, r) \in \mathbb{N}^2$: The simulator simulates q mining queries by the adversary in round r of the simulated execution.

2. Event $I^{\mathtt{A}}_{b,r}$, for each pair $(b, r) \in \mathbb{N}^2$: The simulator inserts b blocks into the state of the ledger in round r, such that all these blocks were previously queries to the (simulated) random oracle by the adversary. Informally, this event occurs when an honest party views these blocks as "confirmed" (part of his own ledger state).

A different payoff is associated with each event. In order to make q mining queries and invoke event $W^{\mathtt{A}}_{q,r}$, the attacker must pay $q \cdot \mathtt{mcost}$, where $\mathtt{mcost}$ is the cost of making a mining query (e.g. electricity cost per hash query). When b blocks made by the adversary are inserted into the ledger and event $I^{\mathtt{A}}_{b,r}$ occurs, the attacker receives payoff $b \cdot \mathtt{breward} \cdot \mathtt{CR}$. Here $\mathtt{breward}$ is the reward for making a block in the currency of the blockchain (e.g. Bitcoins), and $\mathtt{CR}$ is an exchange rate to the same currency used for $\mathtt{mcost}$ (e.g. USD).

Then, [15] defines the following attacker's utility for a strategy profile $(\Pi, \mathcal{A})$. Let $\mathcal{C}_{\mathcal{A}}$ denote the set of simulators that can emulate an adversary $\mathcal{A}$ in the ideal world with access to the weaker ledger functionality $\mathcal{G}^{\hat{\mathbb{B}}}_{\text{WEAK-LEDGER}}$, and $\mathcal{Z}$ denote an environment. The *real payoff* of an adversary $\mathcal{A}$ attacking the protocol is defined as the minimum payoff over all simulators in $\mathcal{C}_{\mathcal{A}}$. If $\mathcal{C}_{\mathcal{A}} = \emptyset$ (there are no simulators that can simulate $\mathcal{A}$) then $u_{\mathtt{A}}(\Pi, \mathcal{A}) = \infty$ by definition. Then, the utility $u_{\mathtt{A}}(\Pi, \mathcal{A})$ is the real payoff, maximized over all possible environments $\mathcal{Z}$ (we assume for simplicity that environments are closed and run in polynomial time in the security parameter [37]).

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} \left(b \cdot \mathtt{breward} \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right) - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \right\} \right\}. \tag{2.2}$$

Strictly speaking, the utilities are also functions in the security parameter κ (the environment obtains the parameter as input in UC) but we omit it for notational simplicity. We note that as functions in the security parameter κ, the asymptotic behavior of the involved functions is the relevant aspect.

2.6.3.2 Utility of the protocol designer $\mathtt{D}$

In [65], the attack game is assumed to be zero-sum, i.e. when considering the designer's utility $u_{\mathtt{D}}$, we set $u_{\mathtt{D}} = -u_{\mathtt{A}}$. To remove this assumption, we follow the methodology of a more recent

work [15] to define $u_{\mathtt{D}}$. In more detail, for each (Π, A), we must assign utility for the designer using the *same* simulators and environments as those used for the attacker. Let $\mathcal{S}_{\mathtt{A}}$ denote the class of simulators that were used to obtain the utility of the attacker, and $\mathcal{Z}_{\mathtt{A}}$ denote the class of environments maximizing the utility for simulators in $\mathcal{S}_{\mathtt{A}}$. That is,

$$\mathcal{S}_{\mathtt{A}} = \left\{ \mathrm{S} \in \mathcal{S}_{\mathrm{A}} \colon \sup_{\mathrm{Z} \in \mathtt{ITM}} \{U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathrm{S}, \mathrm{Z})\} = u_{\mathtt{A}}(\Pi, \mathrm{A}) \right\}$$

and

$$\mathcal{Z}_{\mathtt{A}} = \left\{ \mathrm{Z} \in \mathtt{ITM} \text{: for some } \mathrm{S} \in \mathcal{S}_{\mathtt{A}} \ , \ U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathrm{S}, \mathrm{Z}) = u_{\mathtt{A}}(\Pi, \mathrm{A}) \right\}.$$

Then, let $v_{\mathtt{D}}^{\langle \mathcal{F} \rangle, \mathrm{S}, \mathrm{Z}}$ and $U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathrm{S}, \mathrm{Z})$ be defined similar to the payoffs $v_{\mathtt{A}}^{\langle \mathcal{F} \rangle, \mathrm{S}, \mathrm{Z}}$ and $U_{I^{\mathtt{A}}}^{\langle \mathcal{F} \rangle}(\mathrm{S}, \mathrm{Z})$ respectively. Again following the definitions of [15], the real payoff of the designer is $U_{\mathtt{D}}^{\Pi, \langle \mathcal{F} \rangle}(\mathrm{A}, \mathrm{Z}) = \sup_{\mathrm{S} \in \mathcal{S}_{\mathrm{A}}} \{U_{I^{\mathtt{D}}}^{\langle \mathcal{F} \rangle}(\mathrm{S}, \mathrm{Z})\}$. The utility of the designer is then $u_{\mathtt{D}}(\Pi, \mathrm{A}) := \hat{U}_{\mathtt{D}}^{\Pi, \langle \mathcal{F} \rangle}(\mathrm{A}) = \inf_{\mathrm{Z} \in \mathcal{Z}_{\mathtt{A}}} \{U_{\mathtt{D}}^{\Pi, \langle \mathcal{F} \rangle}(\mathrm{A}, \mathrm{Z})\}$ We can extend the attack model with the value function of the designer $v_{\mathtt{D}}$: $\mathcal{M} = (\mathcal{F}, \langle \mathcal{F} \rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$.

2.6.3.3 Strongly attack-payoff security

This notion of attack-payoff security defined above in Def. 12 does not necessarily mean an incentive-driven adversary will honestly follow the protocol—there is no restriction on the honestly of the actions of $\mathcal{A}'$ in the above definition. To capture this stronger requirement in the context of Bitcoin, we also consider a stronger notion introduced by [15]: the attacker is incentivized to always choose a *front-running, passive-mining* adversary over any (potentially malicious) strategy. Informally, this passive adversary behaves exactly like an honest party (mining with all his hashing power and releasing a block he has found immediately), except the adversary's messages are always delivered before the honest parties' (front-running). Front-running gives the adversary an advantage since if an adversary's block is concurrently competing with an honest party's block to be appended to the longest chain, the adversary always wins.

Definition 15 (Front-running, passive-mining adversary [15])**.** The front-running adversarial strategy $\mathcal{A} \in \mathbb{A}_{\mathtt{fr}}$ is specified as follows: Upon activation in round $r > 0$, $\mathcal{A}$ activates in a round-robin fashion all its (passively) corrupted parties, say $p_1, \ldots, p_t$. When corrupt party p_i generates some new message to be sent through the network, $\mathcal{A}$ immediately delivers it to all its recipients. In addition, upon any activation, any message submitted to the network $\mathcal{F}_{\text{N-MC}}$ by an honest party is maximally delayed.

$\Pi^{\text{฿}}$ was proved to be strongly attack-payoff in [15] for the utility in Equation 2.2. Informally, a protocol is strongly attack-payoff secure if there is always a passive adversarial strategy that is at least as good as any malicious strategy. In our analyses, we are also interested in the case where security does not hold: we say an adversary $\mathcal{A}$ *breaks* strong attack-payoff security if $u_{\mathtt{A}}(\Pi, \mathcal{A})$ exceeds $u_{\mathtt{A}}(\Pi, \mathcal{A}')$ for any $\mathcal{A}' \in \mathbb{A}_{\mathtt{fr}}$, by a non-negligible amount.

Definition 16 (Strongly attack-payoff secure [15])**.** A protocol Π is *strongly attack-payoff secure* for attack model $\mathcal{M}$ if there is a $\mathcal{A}' \in \mathbb{A}_{\mathtt{fr}}$ such that for all $\mathcal{A}$, $u_{\mathtt{A}}(\Pi, \mathcal{A}) \leq u_{\mathtt{A}}(\Pi, \mathcal{A}') + \mathsf{negl}(\kappa)$

In studying the (rational) security of the Bitcoin backbone protocol, we will follow the approach from [15] that simplifies the proofs when analyzing the utilities from mining in the protocol $\Pi^{\Bitcoin}$ by utilizing the composition theorem of RPD. As explained in Section 2.5, instead of analyzing the probabilities of payoff-inducing events for $\Pi^{\Bitcoin}$ which uses the random oracle as the lottery, one can analyze probabilities for the *modular* ledger protocol w.r.t. an idealized lottery that makes use of the state exchange functionality $\mathcal{F}_{\mathrm{StX}}$ (Fig, 2.6). In more detail: when a party (or the adversary in the name of a corrupted party) wishes to extend a chain, they would invoke $\mathcal{F}_{\mathrm{StX}}$ with a SUBMIT-NEW command, which performs a coin toss and informs him whether he is successful. If the party is successful, the functionality includes this new chain into a tree data structure and allows the party to multicast this new chain with a SEND command; this multicasting is done automatically for honest parties. Due to the correspondence of RO queries in the Bitcoin protocol and the SUBMIT-NEW-commands in the modularized Bitcoin protocol [17], the events defined for the utility function in Eqn. 2.2 above remain valid and meaningful also in this hybrid world, because the black-box simulator for the overall Bitcoin protocol simulates one RO-query (as a reaction to an input by a corrupted party) whenever the (black-box) simulator for the modular ledger protocol simulates one SUBMIT-NEW-command, as a reaction to the corresponding input by the same party [15].

Chapter 3

Privacy of Deterministic Voting Rules

3.1 Introduction

Differential privacy (DP) has gained much public attention recently, partly due to its use in the United States 2020 Census. Improving upon ad-hoc privacy techniques that were broken in the previous census [69], formal privacy definition like DP are much more suitable for controlling the leakage of sensitive data.

Yet, sensitive data is still published today without necessarily understanding the privacy leakage it incurs. In particular, voting data has been surprisingly accessible. In the US, histograms of votes are revealed per county, and voting and registration tables are released including fields like sex, race, age, location, and marital status [6]. This abundance of information has enabled politicians to buy voter profiles from data mining companies, allowing them to more easily manipulate public opinion [31, 124].

Unfortunately, differential privacy, though now a standard in data privacy, is not easy to achieve in voting. One must not only protect voter registration tables with proven privacy techniques, but also the election outcome itself. To see how an individual's vote can be inferred by observing the winner of the election, we consider the following example. Suppose *Alice* cast a vote in an election, and then the winner is announced. Further suppose that an adversary can accurately estimate other votes from questionnaires or by machine learning from the other voters' social media, and it turns out these other votes ended up with a tie among the candidates. In this case, the adversary can distinguish Alice's vote even if he knows nothing about Alice, since she must have voted for the winner as the tie-breaker.

The strict definition of differential privacy means the mere *possibility* of the above scenario is a privacy violation. Moreover, ties do occur more often than expected in real life elections; e.g., 9.2% of STV elections on Preflib election data [106] are tied [125]. One can possibly consider another formal privacy definition that accepts the risk of the (arguably unlikely) event of deanonymizing Alice's vote in the above example. For example, it can take into account an attacker's uncertainly on votes in the election stemming from machine learning methods, or low

Most of the results in this chapter have been published in [103].

likelihood of ties. However, even so it is unclear how to quantitatively measure the effect of such uncertainty, and how (or whether) privacy differs for different voting rules.

Motivated by the privacy concern in voting, in this chapter we focus on the following key question.

How private are commonly-used voting rules?

The importance of answering this question is both practical and theoretical. On the practical side, minimizing the amount of an individual voter's information leakage from voting rules helps protect against censorship, coercion, and vote buying. On the theoretical side, privacy provides a new angle to comparing voting rules and designing new ones.

A first attempt would be to employ *differential privacy (DP)* [58] which we introduced formally in Ch. 2, Def. 4. DP is a measure of privacy that has been widely-accepted and widely-applied in the cryptographic community. Below, we re-introduce the definition of DP in the language of voting. Recall that mathematically, a voting rule f^v for $n \in \mathbb{N}$ voters is a mapping $f^v : \mathcal{U}^n \to \mathbf{Range}(f^v)$, where $\mathcal{U}$ is the set of all possible votes; $\mathcal{U}^n$ is a dataset of votes called a *preference profiles* (here, we consider preference profile with n votes); the range of the function f^v, $\mathbf{Range}(f^v)$ is the set of all possible outcomes of voting, e.g. possible winners or histograms/tallies of votes. Then we say f^v is $(\epsilon, \delta)-$*differentially private* if for any pair of preference profiles $\vec{X} \in \mathcal{U}^n$ and $\vec{X}' \in \mathcal{U}^n$ that only differ on one vote, and any subset of outcomes $\mathcal{S} \subseteq \mathbf{Range}(f^v)$, the following inequality holds:

$$\Pr\left[f^v(\vec{X}) \in \mathcal{S}\right] \leq e^{\epsilon} \Pr\left[f^v(\vec{X}') \in \mathcal{S}\right] + \delta. \tag{3.1}$$

Smaller ϵ, δ are desirable as it means the outcome of f^v is affected less by the change of one vote, and thus reveals less about an individual voter. Note in general f^v must be randomized to satisfy Inequality 3.1; indeed [84, 98, 121] achieved DP via randomized voting.

Yet most, if not all, voting rules used in high-stakes political elections are deterministic. Most randomized voting rules suffer from difficulties in verifying implementation correctness—a prominent example being the controversy in the 2016 Democratic primary election in Iowa [48]. In other words, randomized voting can act against voter incentives, by bringing doubt on the legitimacy of the election winner. Unfortunately, the randomness in Inequality 3.1 above comes from the voting rule itself, so deterministic rules cannot achieve DP except with the trivial parameter of $\delta \geq 1$, which always holds. We present the following Example 3 to show more concretely why this is the case.

Example 3 (DP fails for deterministic voting rules)**.** Consider the plurality rule for two candidates $\{a, b\}$ and three voters ($n = 3$). We have $\mathcal{U} = \mathbf{Range}(f^v) = \{a, b\}$. In Inequality 3.1, let $\vec{X} = (a, a, b)$, $\vec{X}' = (b, a, b)$, and $\mathcal{S} = \{a\}$. Then, (Inequality 3.1) becomes $1 \leq e^{\epsilon} \times 0 + \delta$, which means that $\delta \geq 1$.

3.1.1 Overview of our contributions

To overcome the critical limitation of DP in high-stakes voting scenarios, we study the privacy of deterministic voting rules using *distributional differential privacy (DDP)* [22], another well-accepted notion of privacy that works for deterministic functions. Informally, DDP measures the

amount of individual information leakage, while assuming the adversary only has uncertain information about voter preferences—due to e.g., the randomness from a machine learning algorithm. Our result on the DDP of commonly-used voting rules carries the following encouraging message:

Main Message 1: Many commonly-used voting rules achieve good DDP in natural settings.

More precisely, we focus on a natural DDP setting where the adversary's information is represented by a set of i.i.d. distribution's over preference profiles, denoted by $\Delta \subseteq \Pi(\mathcal{U})$, where $\Pi(\mathcal{U})$ is the set of all probability distributions over $\mathcal{U}$ *with full support*. A voting rule f^v's DDP is now measured by three parameters $(\epsilon, \delta, \Delta)$. A deterministic function is DDP (Definition 17) if it satisfies an inequality similar to that in DP, but now the randomness is provided by the adversary's uncertainty about the profile $\vec{X}$, represented by Δ. Like DP, smaller ϵ and δ in DDP are more desirable as they represent *less* information leakage.

With DDP, we can quantitatively measure the privacy of the histogram rule $\mathbf{Hist}$, which outputs the frequency of each type of vote in the preference profile, in the following Theorem 1. As an immediate consequence, many common voting rules also achieve good privacy.

Theorem 1 (DDP for $\mathbf{Hist}$). *Given any $\mathcal{U} = \{x_1, \ldots, x_\ell\}$ and $\Delta \subseteq \Pi(\mathcal{U})$ with $|\Delta| < \infty$, let $p_{\min} = \min_{\pi \in \Delta, i \leq \ell}(\pi(x_i))$. For any $n \in \mathbb{N}$ and any $\epsilon \geq 2\ln\left(1 + \frac{1}{p_{\min} n}\right)$, $\mathbf{Hist}$ for n voters is $(\epsilon, \delta, \Delta)$-DDP where $\delta = \exp(-\Omega(np_{\min}[\min(2\ln(2), \epsilon)]^2))$.*

Theorem 1 states that $\mathbf{Hist}$ is private with good parameters, as even a small ϵ results in δ that is considered *negligible* in cryptography literature. The winner of many commonly-used voting rules depends only on the outcome of $\mathbf{Hist}$, and thus contain (often strictly) less information than $\mathbf{Hist}$. Thus, they achieve *at least as good* privacy w.r.t. DDP as simply outputting the histogram.

Next, we highlight that DDP (as well as DP and its variants) parameters only describe loose bounds on privacy. For example, by definition if a voting rule satisfies $(\epsilon, \delta, \Delta)$-DDP, it also satisfies $(\epsilon + 0.1, \delta + 0.1, \Delta)$-DDP. To compare the privacy-preserving capability of voting rules, we introduce the notion of *exact distributional differential privacy (eDDP)*, whose parameters describe tight bounds on ϵ and δ. We focus on the $\epsilon = 0$ case as a first step to compare various voting rules with their eDDP in the δ parameter. Our results on the eDDP of commonly-used voting rules carry the following message:

Main Message 2: For many combinations of commonly-used voting rules and Δ, the $(0, \delta, \Delta)$-eDDP exhibits a dichotomy between $\delta = \Theta(\sqrt{1/n})$ and $\delta = \exp(-\Omega(n))$.

More precisely, we prove the following dichotomy theorem for two candidates $\{a, b\}$ and α-*biased majority rules* with $\alpha \in (0, 1)$, which chooses a as the winner iff at least αn out of n votes prefer a.

Theorem 2 (Dichotomy in Exact DDP for α-Majority Rules over Two Candidates, Informal) *Fix two candidates $\{a, b\}$ and $\Delta \subseteq \Pi(\{a, b\})$ with $|\Delta| < \infty$. For any $\alpha \in (0, 1)$,*

the α-biased majority rule is $(0, \delta, \Delta)$-eDDP for all n, where δ is either $\Theta(\sqrt{1/n})$, when Δ contains a distribution π with $\pi(a) = \alpha$, or exponentially small otherwise.

For more than two candidates, we prove the following dichotomy theorem for a large family of voting rules and $\Delta \subseteq \Pi(\mathcal{U})$.

Theorem 4 (Dichotomy in Exact DDP of A Large Class of Voting Rules and Δ, Informal) *For any fixed number of candidates, and any voting rule in a large family, the $(0, \delta, \Delta)$-eDDP is $\delta = \Theta(\sqrt{1/n})$, when Δ contains the uniform distribution, or $\delta = \exp(-\Omega(n))$, when Δ is finite and does not contain any unstable distributions.*

Intuitively, a distribution π is *unstable* under a voting rule f^v if adding small perturbations can cause a different candidate to win (Definition 21). Instead of conducting case-by-case studies of eDDP for commonly-used voting rules, we prove Theorem 4 for a large family of voting rules called *generalized scoring rules* [127] that further satisfy *monotonicity, local stability*, and *canceling-out*. We show that many commonly-used voting rules satisfy these conditions (Section 3.4.2). We also compute and compare the concrete δ values for small elections (Table 3.1 below, and Table 3.2 in Section 3.5). Intuitively, 2-approval and plurality are the most private in the list, since votes only reveal the voters' top one or two candidates. Borda, on the other hand, uses the voters' entire rankings to decide the winner, and thus is the least private.

f^v	Borda	STV	Maximin	Plurality	2-approval
$\delta(n)$	$\frac{1}{\sqrt{1.347n + 0.5263}}$	$\frac{1}{\sqrt{1.495n + 0.02669}}$	$\frac{1}{\sqrt{1.553n + 4.433}}$	$\frac{1}{\sqrt{1.717n - 0.09225}}$	$\frac{1}{\sqrt{1.786n + 0.3536}}$

Table 3.1: δ values in $(0, \delta, \Delta)$-eDDP for some commonly-used voting rules under the i.i.d. uniform distribution, $m = 3$ and $n \leq 50$. From left to right, we rank rules from least to most private.

3.1.2 Related works

The first works on DP described how one can create mechanisms for answering standard statistical queries on a database (e.g., number of records with some property or histograms) in a way that satisfies the DP definition. This ignited a vast and rapidly evolving line of research on extending the set of mechanisms and achieving different DP guarantees—we refer the reader to [60] for an (already outdated) survey—to a rich literature of relaxations to the definition, e.g., [21,27,57,101], that capture among others, noiseless versions of privacy, as well as works studying the trade-offs between privacy and utility of various mechanisms [22, 30, 71, 83, 107].

Generalized Scoring Rules (GSRs) is a class of voting rules that include many commonly studied voting rules, such as Plurality, Borda, Copeland, Maximin, and STV [127]. It has been shown that for any GSR the probability for a group of manipulators to be able to change the winner has a phase transition [110, 128]. An axiomatic characterization of GSRs is given in [128]. The most robust GSR with respect to a large class of statistical models has been characterized [42]. Recently GSRs have been extended to an arbitrary decision space, for example to choose a set of winners or rankings over candidates [126].

Relaxations to Differential Privacy and Noiseless Functions and Distributional Differential Privacy (DDP) Relaxations to differential privacy have been proposed to allow functions with less to no noise to achieve a DP-style notion of privacy. Kasiviswanathan and Smith [89] formally proved that differential privacy holds in presence of arbitrary adversarial information, and formulated a Bayesian definition of differential privacy which makes adversarial information explicit. Hall et al. [81] suggested adding noise to only certain values (such as low-count components in histograms) to achieve a relaxed notion of Random Differential Privacy with higher accuracy. Taking advantage of (assumed) inherent randomness in the database, several works have also put forward DP-style definitions which allow for noiseless (i.e., deterministic) functions. Duan [57] showed that sum queries of databases with i.i.d. rows can be outputted without noise. Bhaskar et al. [27] introduced Noiseless Privacy for database distributions with i.i.d. rows, whose parameters depend on how far the query is from a function which only depends on a subset of the database. Motivated by Bayesian mechanism design, Leung and Lui [101], suggested noiseless sum queries and introduced Bayesian differential privacy for database distributions with independent rows, where the auxiliary information is some number of revealed rows.

These ideas were generalized and extended by Bassily et al. who introduced *distributional differential privacy (DDP)* [21, 79]. Informally, given a distribution (X, Z), where X is the adversary's uncertainty in the database distribution and Z is a parameter used for proving composition theorems (i.e. computing DDP when outputting results from two functions that are both DDP with some parameters), we say a function f^v is $(\epsilon, \delta, \Delta = \{(X, Z)\})$-DDP if its output distribution $f^v(X)|Z$ can be simulated by a simulator that is given the database missing one row. In these works, deterministic functions which have been shown to satisfy DDP are exact sums, truncated histograms, and *stable* functions where with large probability, the output is the same given neighboring databases.

3.1.2.1 Differential privacy in rank aggregation

Differential privacy [58] has been used to add privacy to rank aggregation, functions whose inputs are rankings/linear orders (e.g., voting rules aggregate votes, which are rankings on a set of candidates): [121] applied Gaussian noise to the histogram of linear orders, while [84] used Laplace and Exponential mechanisms applied to specific voting rules. [98] also developed a method of random selection of votes to achieve differential privacy. One interesting aspect of adding noise to the output that was observed in [28, 98] is that it enables an approximate strategy-proofness; the idea here is that the added noise dilutes the effect of any individual deviation, thereby making strategies which would slightly perturb the outcome irrelevant. We remark that if one wishes to achieve DP for a large number of voting rules, well-known DP mechanisms (like adding Laplace noise [59]) can be applied to rules in GSR in a straightforward way, by adding noise to each component of the score vector and outputting the winner based on the noised score vector. In contrast, our results focus on the exact privacy of deterministic voting rules, which are more commonly used.

In this chapter, we compare deterministic functions by their exact privacy. In differential privacy literature where functions must be randomized, their accuracy, or utility, is often used to compare them. A number of works have defined utility as a metric which describes the comparative

desirability of ϵ-DP mechanisms. In [107], utility is an arbitrary user-defined function, used in the exponential mechanism. The works of [22, 30, 83] define utility in terms of error, where the closer (by some metric) the output of the function, which uses this mechanism to apply noise, is from the desired (deterministic) query's, the higher the utility; the definition of [71] in addition allows the user to define as a parameter, the prior distribution on the query output. In contrast, our results imply that in the context of distributional differential privacy, voting rules achieve a well-accepted notion of privacy while preserving *perfect* accuracy/utility.

3.1.3 Summary of contributions

To the best of our knowledge, we are the first to illustrate how to measure privacy in high-stakes voting using (e)DDP in a natural setting. We will see that the problem, though challenging, can be solved by our novel *trails* technique. Below we summarise our conceptual and technical contributions.

Conceptual overview. Our first conceptual contribution is the application of DDP to deterministic voting rules. We note that the *truncated* histogram result of [21] does not suffice, since in general, votes are not removed in an election. Moreover, we prove our results in a simpler definition than DDP; the equivalence of this definition and DDP is proven in Lemma 1. Our second conceptual contribution is the introduction of *exact DDP*, addressing the issue that parameters of DDP (and other relaxations of DP [21, 27, 57, 79, 81, 89]) describe only upper bounds on privacy. We are not aware of other works that explicitly propose to characterize tight bounds on the privacy parameters ϵ and δ.

Technical contributions. Our first theorem (Theorem 1) is quite positive, showing the privacy of outputting histograms. Theorems 2, 3, and 4 characterize eDDP in terms of δ values by fixing $\epsilon = 0$. We do so for the two reasons: (1) it is the common convention to compute δ based on a fixed ϵ for DP or DDP; (2) $\epsilon = 0$ is the most informative choice, since Theorem 1 shows that even for small non-zero ϵ, any difference we can observe in the δ of two voting rules is exponentially small—considered negligible in cryptography literature. While our theorems appear similar and related to the dichotomy theorems on the probability of ties in voting [126, 127], the definition and mathematical analysis are quite different, and previous techniques do not work for all cases; see more discussions in the proof sketch for Theorem 4. To address the challenge, we developed the *trails* technique, which significantly simplifies calculations. Informally, the trails technique generalises the intuitive idea that the privacy of voting rules depends on the probability of a tie in the voting outcome.

Generality of our setting. As the first work towards answering our key question, we assume the adversary's beliefs are modeled by a set of i.i.d. distributions over the votes. A special case is the i.i.d. uniform distribution, which is known as the *impartial culture* assumption in social choice [80]. Extending to general (ϵ, δ), and non-i.i.d. distributions is an important and challenging future

direction. Lastly, though our definitions and results are presented in the context of voting for the sake of presentation, they can easily be extended to general applications.

Chapter organization. In Section 3.2, we introduce the version of the DDP definition which we will use in all our analyses. We prove the privacy of outputting a histogram of votes, and as corollary, show that all *anonymous* voting rules—those that depend only on the histogram of votes—also achieve this arguably good privacy. In Section 3.3, we introduce the notion of *exact* DDP (eDDP), in order to compare the privacy of different voting rules. We prove the eDDP of a class of two-candidate voting rules, using this as a demonstration of our new *trails* proof technique. In Section 3.4, we move on to voting rules with an arbitrary number of candidates. We first establish the eDDP of the histogram function, which intuitively informs us of the "worst case" privacy for all anonymous voting rules. Then, we prove the eDDP of a large subset of *generalized scoring rules (GSRs)*, showing the existence of an asymptotic separation of privacy among voting rules. Finally, in Section 3.5, we directly compute of privacy for various voting rules for a limited number of voters. This calculation both confirms our theoretical results, and allows us to rank the privacy of these voting rules using concrete numbers.

3.2 Distributional differential privacy for voting

Our use of distributional differential privacy (DDP) is motivated by the unsuitability of differential privacy (DP) to analyze deterministic voting rules. In the rest of this chapter, we will discuss our definitions and results with respect to voting, although they could be generalized to other applications.

At a high level, the DDP of a (deterministic or randomized) function is characterized by three parameters $(\epsilon, \delta, \Delta)$, where ϵ and δ are privacy parameters similar to DP, and Δ is a set describing the adversary's knowledge about the preference profile. To simplify presentation, below we will introduce a simpler alternative definition of DDP which we will use in our analyses. While this alternative definition is not in general equivalent to DDP, it is equivalent under our settings. The difference between this simper version (Def. 17) and DDP (Def. 5) is that we do not use the"simulator" function h, and the definition also omits the "auxiliary information" Z in Δ. Roughly, Z encodes the additional information leakage from composing multiple functions, an aspect we do not consider as our goal is to compare voting rules. Instead, we encode adversarial knowledge of the votes using $\Delta \subseteq \Pi(\mathcal{U})$, i.e. a set of preference profile distributions where each vote is i.i.d..

Definition 17 (DDP studied in this chapter)**.** For any $\Delta \subseteq \Pi(\mathcal{U})$, $\epsilon > 0$, and $\delta > 0$, a voting rule f^v is $(\epsilon, \delta, \Delta)$-DDP if for every $\pi \in \Delta$, $i \leq n$, $x, x' \in \mathcal{U}$, and $\mathcal{S} \subseteq \mathbf{Range}(f^v)$, the following inequality holds.

$$\Pr_{\vec{X}\sim\pi}(f^v(\vec{X}) \in \mathcal{S} | \vec{X}_i = x) \leq e^\epsilon \Pr_{\vec{X}\sim\pi}(f^v(\vec{X}) \in \mathcal{S} | \vec{X}_i = x') + \delta, \tag{3.2}$$

where $\vec{X}$ is a preference profile of n votes where each vote is i.i.d. generated from π.

Lemma 1 (Equivalence of definitions)**.** *We denote the DDP defined in [21] by* simulation-based *DDP*[1]*. For any $\mathcal{U}$, let $\Delta \subseteq \Pi(\mathcal{U})$ and $\Delta' = (\Delta, Z = \emptyset)$ (where Z is a parameter in the [21] definition). Suppose f^v is $(\epsilon, \delta, \Delta')$-simulation-based DDP, then f^v is $(2\epsilon, (1+e^\epsilon)\delta, \Delta)$-DDP for our Definition 17. Conversely, if f^v is $(\epsilon, \delta, \Delta)$-DDP for Definition 17 then f^v satisfies $(\epsilon, \delta, \Delta')$-simulation-based DDP.*

Proof. We prove the first statement, that is, if f^v is $(\epsilon, \delta, \Delta')$-simulation-based DDP [21], then f^v is $(2\epsilon, (1+e^\epsilon)\delta, \Delta)$-DDP of Definition 17.

By the definition of f^v being $(\epsilon, \delta, \Delta')$-simulation-based DDP, the function h has to satisfy the below inequalities for any $(\pi, Z) \in \Delta'$, any i, and $x \in \mathbf{Supp}(\vec{X}_i)$ (for $\vec{X} \sim \pi$). With $Z = \emptyset$, we can write the inequalities in the DDP definition without Z as

$$\Pr_{\vec{X}\sim\pi}(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x) \leq e^\epsilon \Pr_{\vec{X}\sim\pi}(h(\vec{X}_{-i}) \in \mathcal{S} \mid \vec{X}_i = x) + \delta$$

$$\Pr_{\vec{X}\sim\pi}(h(\vec{X}_{-i}) \in \mathcal{S} | \vec{X}_i = x) \leq e^\epsilon \Pr_{\vec{X}\sim\pi}(f^v(\vec{X}) \in \mathcal{S} | \vec{X}_i = x) + \delta \tag{3.3}$$

(We make $\vec{X} \sim \pi$ implicit to ease presentation.) Now consider any $x' \in \mathbf{Supp}(\vec{X}_i)$, possibly different from the x above. By the definition of DDP, the inequalities should also hold for x', i.e.

$$\Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^\epsilon \Pr(h(\vec{X}_{-i}) \mid \vec{X}_i = x') + \delta$$

Since h is not given ith entry of the database, its output does not depend on the value of the ith row. Moreover, if database rows are independent, the distributions $\vec{X}_{-i}|\vec{X}_i = x' = \vec{X}_{-i}|\vec{X}_i = x$. Thus $\Pr(h(\vec{X}_{-i}) \mid \vec{X}_i = x') = \Pr(h(\vec{X}_{-i}) \in \mathcal{S} \mid \vec{X}_i = x)$. So,

$$\begin{aligned}
&\Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^\epsilon \Pr(h(\vec{X}_{-i}) \in \mathcal{S} \mid \vec{X}_i = x) + \delta \\
&\Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^\epsilon (e^\epsilon \Pr(f^v(\vec{X}) \in \mathcal{S} | \vec{X}_i = x) + \delta) + \delta \text{ (By Equation 3.3 above.)} \\
&\Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^{2\epsilon} \Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x) + e^\epsilon\delta + \delta
\end{aligned}$$

Thus, we have shown that for all $x, x' \in \mathbf{Supp}(\vec{X}_i)$ (and all i),

$$\Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^{2\epsilon} \Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x) + (e^\epsilon + 1)\delta$$

So, f^v is $(2\epsilon, (1+e^\epsilon)\delta, \Delta)$-DDP, proving the first statement.

We now prove the second statement. That is, if f^v is $(\epsilon, \delta, \Delta)$-DDP of Definition 17 then f^v is $(\epsilon, \delta, \Delta')$-simulation-based DDP. To do so, we define the function h to be the algorithm which inserts any $x' \in \mathbf{Supp}(\vec{X}_i)$ to the missing ith row of the database, and apply f^v to the result.

[1](Following the function h being referred to as the "simulator" in [21])

By independence of rows, $\Pr(h(\vec{X}_{-i}) \mid \vec{X}_i = x) = \Pr(h(\vec{X}_{-i}) \mid \vec{X}_i = x')$ by our definition of h, equal to $\Pr(f^v(\vec{X}) \mid \vec{X}_i = x')$. Then, for any $\vec{X} \in \Delta$, i, and $x, x' \in \mathbf{Supp}(\vec{X}_i)$,

$$\Pr(h(\vec{X}_{-i}) \in \mathcal{S} \mid \vec{X}_i = x) = \Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x') \leq e^{\epsilon} \Pr(f^v(\vec{X}) \in \mathcal{S} \mid \vec{X}_i = x) + \delta$$

by inequality of Definition 17. This proves the second statement.

□

3.2.1 Privacy of the histogram function

We show that the histogram function $\mathbf{Hist}$ (Def. 6) satisfies good DDP. As corollary, all voting rules (whether deterministic or randomized) whose output only depends on the histogram of votes also satisfy good DDP. We will show this corollary by proving that DDP (both the definition of [21] and our simplification) is immune to *post-processing* (Lem. 2), a notion originally defined for DP.

Theorem 1 (DDP of $\mathbf{Hist}$)**.** *Given any $k \in \mathbb{N}$. $\mathcal{U} = \{x_1, \ldots, x_\ell\}$ and $\Delta \subseteq \Pi(\mathcal{U})$ with $|\Delta| < \infty$, let $p_{\min} = \min_{\pi \in \Delta, i \leq \ell}(\pi(x_i))$. Then $\mathbf{Hist}$ is $(\epsilon(n), \delta(n), \Delta)$-DDP where $\epsilon(n) = 2\ln(1 + \frac{1}{p_{\min} n})$ and $\delta(n) = \exp(-\Omega(np_{\min}[\min(2\ln(2), \epsilon)]^2))$, where n is the number of votes in the preference profile.*

Proof. At a high level, the proof is similar to Theorem 8 of [101].

Fix $\pi \in \Delta$. Since votes are i.i.d. and all $i \in [n]$ are equivalent for the histogram, we simplify $\Pr_{\vec{X} \sim \pi}(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x)$ as $\Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{S})$, where $\vec{X}_{-1}$ refers to $\vec{X}$ without the first vote.

We need to show that for all $x_i, x_j \in \{x_1, \cdots, x_\ell\}$, and all $\mathcal{S} \subseteq \mathbb{N}^\ell$:

$$\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{S}) \leq e^{\epsilon} \Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) \in \mathcal{S}) + \delta$$

We observe that for any set $\mathcal{B}$ and x:

$$\Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{S}) = \Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}}) + \Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{S} \cap \mathcal{B}) \quad (3.4)$$

$$\leq \Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}}) + \Pr(\mathbf{Hist}(x, \vec{X}_{-1}) \in \mathcal{B}) \quad (3.5)$$

Let $\mathcal{B}$ be the set of all histogram $t \in \mathbb{N}^\ell$ where $t_i > p_i(n-1)e^{\epsilon/2}$ and $t_j < p_j(n-1)e^{-\epsilon/2}$. Fix a choice of $\epsilon > 2\ln(1 + \frac{1}{p_{\min} n})$. We claim that for $\delta = \exp(\Omega(np_{\min}(\min(2\ln(2), \epsilon))^2)$, the following hold:

Claim 1: $\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}}) \leq e^{\epsilon} \Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}})$

Claim 2: $\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{B}) \leq \delta$

If both claims are true, then by Inequality (3.5),

$$\begin{aligned}\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{S}) &\leq \Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}}) + \Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{B}) \\ &\leq e^{\epsilon} \Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) \in \mathcal{S} \cap \overline{\mathcal{B}}) + \delta \\ &\leq e^{\epsilon} \Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) \in \mathcal{S}) + \delta\end{aligned}$$

which proves the theorem. Below we will prove both claims.

Claim 1 proof:
Since all entries in random variable $\vec{X}_{-1}$ are i.i.d., the random variable $\mathbf{Hist}(\vec{X}_{-1})$ which outputs the histogram of the database has distribution equal to the multinomial distribution on $n-1$ trials and ℓ events:

$$\Pr(\mathbf{Hist}(\vec{X}_{-1}) = (t_1, \cdots, t_\ell)) = \frac{(n-1)!}{t_1! \cdots t_\ell!}\, p_1^{t_1} \cdots p_\ell^{t_\ell}$$

where t_i is the count of entries with the value x_i and p_i is the probability for an entry to have the value x_i.

Thus,

$$\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) = (t_1, \cdots, t_\ell)) = \frac{(n-1)!}{t_1! \cdots (t_i - 1)! \cdots t_\ell!} p_1^{t_1} \cdots p_i^{t_i - 1} \cdots p_\ell^{t_\ell}$$

and

$$\Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) = (t_1, \cdots, t_\ell)) = \frac{(n-1)!}{t_1! \cdots (t_j - 1)! \cdots t_\ell!} p_1^{t_1} \cdots p_i^{t_j - 1} \cdots p_\ell^{t_\ell}$$

So, for every $t = (t_1, \cdots, t_\ell) \in \mathcal{S} \cap \overline{\mathcal{B}}$:

$$\begin{aligned}\frac{Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) = t)}{Pr(\mathbf{Hist}(x_j, \vec{X}_{-1}) = t)} &= \frac{\frac{(n-1)!}{t_1! \cdots (t_i - 1)! \cdots t_\ell!} p_1^{t_1} \cdots p_i^{t_i - 1} \cdots p_\ell^{t_\ell}}{\frac{(n-1)!}{t_1! \cdots (t_j - 1)! \cdots t_\ell!} p_1^{t_1} \cdots p_i^{t_j - 1} \cdots p_\ell^{t_\ell}} \\ &= \frac{t_i}{p_i} \cdot \frac{p_j}{t_j} \\ &= \frac{t_i}{p_i(n-1)} \cdot \frac{p_j(n-1)}{t_j} \\ &\quad \text{By definition of } \mathcal{B},\ t_i > p_i(n-1)e^{\epsilon/2} \text{ or } t_j < p_j(n-1)e^{-\epsilon/2}, \\ &\quad\quad \text{so } t \in \overline{\mathcal{B}} \text{ has } t_i \leq t_i(n-1)e^{\epsilon/2} \text{ and } t_j \geq p_j(n-1)e^{-\epsilon/2} \\ &\leq \frac{p_i(n-1)e^{\epsilon/2}}{p_i(n-1)} \frac{p_j(n-1)}{p_j(n-1)e^{-\epsilon/2}} = e^{\epsilon/2} \times e^{\epsilon/2} = e^{\epsilon}\end{aligned}$$

This proves Claim 1.

Claim 2 proof: Recall $\mathcal{B}$ is the set of all histogram $t \in \mathbb{N}^\ell$ where $t_i > p_i(n-1)e^{\epsilon/2}$ and $t_j < p_j(n-1)e^{-\epsilon/2}$. For any $i \in \{1, \cdots, \ell\}$ let $\mathbf{Hist}(x, \vec{X}_{-1})_i$ denote ith component of the random variable $\mathbf{Hist}(x, \vec{X}_{-1})$.

$$\begin{aligned}
&\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{B}) \\
&\quad = \Pr\left(\mathbf{Hist}(x_i, \vec{X}_{-1})_i > p_i(n-1)e^{\epsilon/2} \text{ or } \mathbf{Hist}(x_i, \vec{X}_{-1})_j < p_j(n-1)e^{-\epsilon/2}\right) \\
&\quad \le \Pr\left(\mathbf{Hist}(x_i, \vec{X}_{-1})_i > p_i(n-1)e^{\epsilon/2}\right) + \Pr\left(\mathbf{Hist}(x_i, \vec{X}_{-1})_j < p_j(n-1)e^{-\epsilon/2}\right) \\
&\qquad\qquad \text{(By union bound)} \\
&\quad = \Pr\left(1 + \mathbf{Bin}(n-1, p_i) > p_i(n-1)e^{\epsilon/2}\right) + Pr\left(\mathbf{Bin}(n-1, p_j) < p_j(n-1)e^{-\epsilon/2}\right) \\
&\qquad\qquad \text{(Where } \mathbf{Bin}(n,p) \text{ denotes binomial r.v. with } n \text{ trials and success probability } p) \\
&\quad = \Pr(\mathbf{Bin}(n-1, p_i) > p_i(n-1)(e^{\epsilon/2} - (p_i(n-1))^{-1})) \\
&\qquad + \Pr(\mathbf{Bin}(n-1, p_j) < p_j(n-1)e^{-\epsilon/2})
\end{aligned}$$

The random variable $\mathbf{Bin}(n-1, p_i)$ has mean $\mu = p_i(n-1)$. When

$$2\ln(1 + \frac{1}{p_i(n-1)}) < 2\ln(1 + \frac{1}{p_{\min}(n-1)}) < \epsilon \le 2\ln(2) < 2\ln(2 + \frac{1}{p_i(n-1)})$$

we have $0 < \beta = e^{\epsilon/2} - (p_i(n-1))^{-1} - 1 < 1$. By Chernoff bound,

$$\begin{aligned}
\Pr(\mathbf{Bin}(n-1, p_i) > (1+\beta)\mu &\le e^{-\mu\beta^2/3} \\
&= \exp(-\Omega(p_i(n-1)(e^{\epsilon/2} - (p_i(n-1))^{-1} - 1)^2)) \\
&= \exp(-\Omega(p_i n\epsilon^2))
\end{aligned}$$

The random variable $\mathbf{Bin}(n-1, p_j)$ has mean $\mu = p_j(n-1)$. By Chernoff bound, for any $0 < \beta = 1 - e^{-\epsilon/2} < 1$ (ie. $\epsilon > 0$),

$$\begin{aligned}
Pr(\mathbf{Bin}(n-1, p_j) < (1-\beta)\mu) &\le e^{-\mu\beta^2/2} \\
&= \exp(-\Omega(p_j(n-1)(1 - e^{-\epsilon/2})^2)) \\
&= \exp(-\Omega(p_j n\epsilon^2))
\end{aligned}$$

So that:

$$\begin{aligned}
\Pr(\mathbf{Hist}(x_i, \vec{X}_{-1}) \in \mathcal{B}) &\le \Pr(\mathbf{Bin}(n-1, p_i) > p_i(n-1)(e^{\epsilon/2} - (p_i(n-1))^{-1})) \\
&\quad + \Pr(\mathbf{Bin}(n-1, p_j) < p_j(n-1)e^{-\epsilon/2}) \\
&\le \exp(-\Omega(p_i n\epsilon^2)) + \exp(-\Omega(p_j n\epsilon^2)) \\
&\le \exp(-\Omega(p_{\min} n\epsilon^2)) = \delta
\end{aligned}$$

for $2\ln(1+\frac{1}{p_{\min}(n-1)}) < \epsilon \leq 2\ln(2)$. To get rid of the upper bound on ϵ, notice when $\epsilon = 2\ln(2)$, $\delta = \exp(-\Omega(p_{\min} n (2\ln(2))^2))$ suffices to satisfy the inequality

$$\Pr(\mathbf{Hist}(x_i, X_{-1}) \in \mathcal{S}) \leq e^{\epsilon} \Pr(\mathbf{Hist}(x_j, X_{-1}) \in \mathcal{S}) + \delta$$

Thus, when $\epsilon > 2\ln(2)$, the same $\delta = \exp(\Omega(np_{\min}[\min(2\ln(2), \epsilon)]^2)) = \exp(-\Omega(np_{\min}(2\ln(2))^2))$ also suffices, as a larger ϵ only makes the right hand side of the inequality larger.

This proves Claim 2.

□

3.2.1.1 Corollary for voting rules

The privacy of the histogram function $\mathbf{Hist}$ implies the privacy of voting rules which depend on the histogram of votes—i.e., anonymous voting rules, where the identity of the voter does not matter—are also private. This follows from the fact that similar to differential privacy, DDP is immune to *post-processing*. Specifically, we prove that composing *any* $(\epsilon, \delta, \Delta)$-DDP f^v with a deterministic function g (as we focus on deterministic voting rules) results in a $(\epsilon, \delta, \Delta)$-DDP function.

Looking ahead, we note that immunity to post-processing is not a property of *exact privacy* which we define in the next section, since exact privacy describes tight bounds on ϵ, δ.

Lemma 2 (Immunity to post-processing)**.** *Suppose $f : \mathcal{U}^* \to \mathcal{O}$ is $(\epsilon, \delta, \Delta)$-DDP. Let $g : \mathcal{O} \to \mathcal{O}'$ be a deterministic function. Then $g \circ f : \mathcal{U}^* \to \mathcal{O}'$ is also $(\epsilon, \delta, \Delta)$-DDP.*

Proof. For any $\pi \in \Delta$, $x, x' \in \mathbf{Supp}(\vec{X}_i)$ and $\mathcal{S} \subseteq \mathcal{O}'$, let $\mathcal{W} = \{w \ : \ g(w) \in \mathcal{S}\}$. Then

$$\begin{aligned}
&\Pr(g(f(\vec{X})) \in \mathcal{S} \mid \vec{X}_i = x) \\
&= \Pr(f(\vec{X}) \in \mathcal{W} \mid \vec{X}_i = x) && \text{(By definition of } \mathcal{W}) \\
&\leq e^{\epsilon} \Pr(f(\vec{X}) \in \mathcal{W} \mid \vec{X}_i = x') + \delta && \text{(By } f \text{ being } (\epsilon, \delta, \Delta)\text{-DDP)} \\
&= e^{\epsilon} \Pr(g(f(\vec{X})) \in \mathcal{S} \mid \vec{X}_i = x) + \delta && \text{(By definition of } \mathcal{W})
\end{aligned}$$

□

Corollary 1. *We call a voting rule* anonymous *if its output does not depend on the identity of the voter. All anonymous voting rules are $(\epsilon(n), \delta(n), \Delta)$-DDP where $\Delta \subseteq \Pi(\mathcal{U})$, $\epsilon(n) = 2\ln(1+\frac{1}{p_{\min} n})$ and $\delta(n) = \exp(-\Omega(np_{\min}[\min(2\ln(2), \epsilon)]^2))$, where n is the number of votes in the preference profile.*

3.3 The exact privacy of voting rules: two candidates

As we briefly discussed in the introduction Section 3.1.1, the definitions of DDP (and DP) are unsuitable for *comparing* the privacy of voting rules, as they describe only a possibly loose

bound on privacy. To resolve this, in this section we present the definition of *exact distributional differential privacy* (exact DDP or eDDP). We will then introduce a technique called the *trails* technique which will in the next section aid us in analyzing exact privacy of general voting rules. As a toy application of this technique, we characterize $(0, \delta, \Delta)$-eDDP for two candidates under any α-biased majority rule.

Intuitively, a function has *exact privacy* with parameters ϵ and δ if it cannot be private with strictly better parameters. We remark that this definition can easily be altered to define (ϵ, δ)-*exact differential privacy (eDP)* by omitting Δ.

Definition 18 (Exact distributional differential privacy (eDDP))**.** A function f is $(\epsilon, \delta, \Delta)$-*exact distributional differentially private (eDDP)* if it is $(\epsilon, \delta, \Delta)$-DDP and there does not exist $(\epsilon' \leq \epsilon, \delta' < \delta)$ nor $(\epsilon' < \epsilon, \delta' \leq \delta)$ such that f is $(\epsilon', \delta', \Delta)$-DDP.

The α-*biased majority rule*, denoted by f^v_α, over two candidates (a, b) outputs a as the winner if at least α fraction of votes prefer a over b. The most simple example of this type of voting rule is *plurality* (Sec. 2.3) with two candidates, which is an $\alpha = 0.5$-biased majority rule. Another example is *supermajority* with $\alpha > 0.5$, which has been used in government decisions around the world. We categorize the exact privacy of α-biased majority rules in Theorem 2, which we will prove using the trails technique we present in the next subsection.

3.3.1 Our tool to analyze privacy: trails technique

Let us describe the trails technique using a simple, toy example: suppose there are two candidates $\{a, b\}$, and $n = 5$ votes. Let $f^v_{0.5}$ be the majority (plurality) rule where ties are broken in favor of a, i.e. $\alpha = 0.5$. We want to compute $(0, \delta, \Delta)$-eDDP of $f^v_{0.5}$ for any $\Delta \subseteq \Pi(\{a, b\})$. In light of Definitions 17 and 18, we have:

$$\delta = \max_{\mathcal{S}, x, x', i, \pi \in \Delta} \left[\Pr_{\vec{X} \sim \pi}(f^v_{0.5}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x) - \Pr_{\vec{X} \sim \pi}(f^v_{0.5}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x') \right]. \tag{3.6}$$

Now, the majority rule is *anonymous*, that is, the identity of the voter is irrelevant and it chooses the winner only based on the histogram of votes (i.e. how many people voted for whom). We can thus write $f^v_{0.5} = f \circ \mathbf{Hist}$, where $t = (t_a, t_b)$ and $f(t)$ outputs a if $t_a \geq t_b$ and outputs b otherwise. Then, Equation (3.6) can be rewritten with probabilities over histograms, which is easier to compute (below, $\vec{X} \sim \pi$ is implicit).

$$\begin{aligned} \delta &= \max_{\mathcal{S}, x, x', i, \pi \in \Delta} \left[\Pr(f(\mathbf{Hist}(\vec{X})) \in \mathcal{S} | \vec{X}_i = x) - \Pr(f(\mathbf{Hist}(\vec{X})) \in \mathcal{S} | \vec{X}_i = x') \right] \\ &= \max_{\mathcal{S}, x, x', i, \pi \in \Delta} \left[\sum_{t \colon f(t) \in \mathcal{S}} \Pr(\mathbf{Hist}(\vec{X}) = t | \vec{X}_i = x) - \sum_{t \colon f(t) \in \mathcal{S}} \Pr(\mathbf{Hist}(\vec{X}) = t | \vec{X}_i = x') \right]. \end{aligned} \tag{3.7}$$

Intuitively, we define *trails* as follows: if $\mathcal{S} = \{a\}$, then $\mathsf{T} \equiv \{t \colon f(t) \in \mathcal{S}\} = \{(5, 0), (4, 1), (3, 2)\}$ is an example of a trail. Intuitively, a trail T is a set of histograms (outputs

of $\mathbf{Hist}$ function) which are *consecutive* in the sense that, starting from some histogram t, we can list exactly the elements of T by iteratively subtracting 1 from and adding 1 to two components of t, respectively. In our example, this corresponds to iteratively changing one vote for a to a vote for b. We see that T can be listed in such a way, starting from entry $\mathbf{Enter}(\mathsf{T}) = (5,0)$ and ending at exit $\mathbf{Exit}(\mathsf{T}) = (3,2)$, by iteratively subtracting from the 1st component and adding to the 2nd component of $(5,0)$ (thus we say the *direction* of T is $(1,2)$). We denote by $\mathbf{End}(a)$ (which we will define formally and use extensively in the proof of Theorem 4) the set of histograms which are exits of the longest trail, along any direction, where a is the winner in all histograms in the trail. In this example $\mathbf{End}(a) = \{(3,2)\}$. See Figure 3.1.

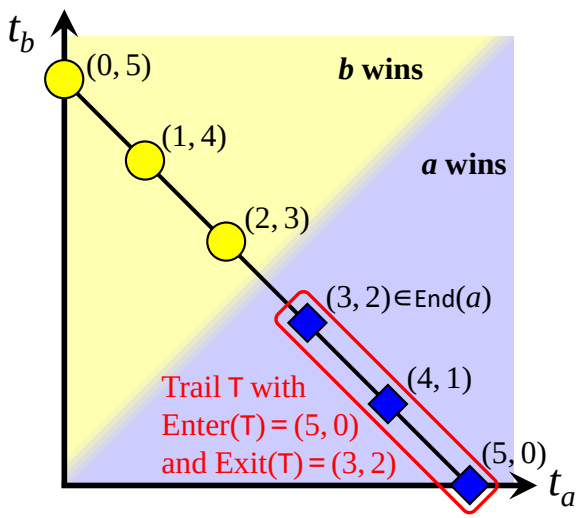

Figure 3.1: A trail for two candidates. A graph of number of votes for candidate a $(= t_a)$ versus votes for candidate b $(= t_b)$. Each point in the line is a histogram where the total number of votes is $n = 5$. The set $\{(5,0),(4,1),(3,2)\}$ forms a trail.

We now give intuition for our key Lemma 3 presented below using this example. Suppose in Equation (3.7) the maximizing $\mathcal{S}$ is $\{a\}$ (so that $\{t\colon f(t) \in \mathcal{S}\} = \mathsf{T}$), $x = a$, and $x' = b$. Then, for any i, and any $\pi \in \Delta$:

$$\delta = \sum_{t \in \{(5,0),(4,1),(3,2)\}} \Pr(\mathbf{Hist}(\vec{X}) = t | \vec{X}_i = a) - \sum_{t \in \{(5,0),(4,1),(3,2)\}} \Pr(\mathbf{Hist}(\vec{X}) = t |\ \vec{X}_i = b).$$

The core of Lemma 3 is the observation that when votes are independent (e.g. when $\Delta \subseteq \Pi(\{a,b\})$), then for all $t = (t_a, t_b)$ such that $t_a > 0$, the following holds:

$$\Pr(\mathbf{Hist}(\vec{X}) = (t_a, t_b) | \vec{X}_i = a) = \Pr(\mathbf{Hist}(\vec{X}) = (t_a - 1, t_b + 1) | \vec{X}_i = b).$$

In light of this, $\Pr(\mathbf{Hist}(\vec{X}) = (5,0) | \vec{X}_i = a)$ cancels out with $\Pr(\mathbf{Hist}(\vec{X}) = (4,1) | \vec{X}_i = b)$, and $\Pr(\mathbf{Hist}(\vec{X}) = (4,1) | \vec{X}_i = a)$ cancels out with $\Pr(\mathbf{Hist}(\vec{X}) = (3,2) | \vec{X}_i = b)$. This leaves:

$$\delta = \Pr(\mathbf{Hist}(\vec{X}) = (3,2) = \mathbf{Exit}(\mathsf{T}) = \mathbf{End}(a) | \vec{X}_i = a) - \Pr(\mathbf{Hist}(\vec{X}) = (5,0) = \mathbf{Enter}(\mathsf{T}) | \vec{X}_i = b). \tag{3.8}$$

We note that here $\Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}) | \vec{X}_i = b) = 0$, but this does not hold generally for all trails for $m \geq 2$. This calculation can be extended to the more general Lemma 3 below. Before that, let us formally define trails. For any histogram $t = (t_1, \cdots, t_\ell) \in \mathbb{N}^\ell$, any $z \in \mathbb{Z}$ and $j \leq \ell$, we let $(t_1, \cdots, t_\ell) + zx_j$ denote the histogram $(t_1, \cdots, t_j + z, \cdots t_\ell)$.

Definition 19 (Trails)**.** Given a pair of indices (j,k) where $j \neq k$, a histogram t, and a length q, we define the *trail* $\mathsf{T}_{t,x_j,x_k,q} = \{t - zx_j + zx_k) : 0 \leq z \leq q\}$, where (j,k) is called the direction of the trail, t is then the *entry* of this trail, also denoted by $\mathbf{Enter}(\mathsf{T}_{t,x_j,x_k,q})$, and $t - qx_j + qx_k$ is called the *exit* of the trail, denoted by $\mathbf{Exit}(\mathsf{T}_{t,x_j,x_k,q})$.

Alternatively, a trail T can be defined by just its entry and exit and the direction can be inferred.

Note on $\mathbf{End}(\cdot)$ and trails defined on profiles: For voting rules where the identity of voters does not matter (i.e. anonymous voting rules, such as the generalized scoring rules (GSR) which we study), the profile of votes is equivalent to the histogram of votes, and thus a trail can simply be defined on profiles as well. Here, we only give a intuitive idea of the relationship between $\mathbf{End}(\cdot)$ and δ via Eq. 3.8. We define $\mathbf{End}(\cdot)$ formally in Definition 23 with respect to trails defined on profiles, where it is relevant in the proof of Theorem 4.

Lemma 3. *Let T be a trail with direction (j,k), and let $\pi \in \Pi(\mathcal{U})$. For any i, $x_j, x_k \in \mathcal{U}$, we have:*

$$\begin{aligned} &\Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathsf{T} \mid \vec{X}_i = x_j) - \Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathsf{T} \mid \vec{X}_i = x_k) \\ = &\Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}) \mid \vec{X}_i = x_j) - \Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}) \mid \vec{X}_i = x_k). \end{aligned}$$

Proof. Fix distribution π over n votes, where each vote is independently distributed. For $\vec{X} \sim \pi$, we recall that we denote $\vec{X}_{-i}$ as the random variable $\vec{X}$ but without the ith vote. The equality in the lemma comes from the simple observation that when votes are independently distributed, for any histogram $t \in \mathbb{N}^\ell$ and any $j \in [\ell]$

$$\Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) = t | \vec{X}_i = x_j) = \Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}_{-i}) = t - x_j)$$

(Below, $\vec{X} \sim \pi$ is implicit). Let q be the length of the trail. For any $0 \leq z < q$, let $t_z = \mathbf{Enter}(\mathsf{T}) - zx_j + zx_k$. Then,

$$\begin{aligned} &\Pr(\mathbf{Hist}(\vec{X}) = t_z | \vec{X}_i = x_j) \\ &= \Pr(\mathbf{Hist}(\vec{X}_{-i}) = t_z - x_j) \\ &= \Pr(\mathbf{Hist}(\vec{X}) = t_z - x_j + x_k | \vec{X}_i = x_k) \\ &= \Pr(\mathbf{Hist}(\vec{X}) = t_{z+1} | \vec{X}_i = x_k). \end{aligned}$$

In other words,

$$\begin{aligned}
&\Pr(\mathbf{Hist}(\vec{X}) \in \mathsf{T} | \vec{X}_i = x_j) - \Pr(\mathbf{Hist}(\vec{X}) \in \mathsf{T} | \vec{X}_i = x_k) \\
=&\Pr(\mathbf{Hist}(\vec{X}) = t_q | \vec{X}_i = x_j) \\
&- \Pr(\mathbf{Hist}(\vec{X}) = t_0 | \vec{X}_i = x_k) \\
&+ \sum_{0 \le z < q} \Big(\Pr(\mathbf{Hist}(\vec{X}) = t_z | \vec{X}_i = x_j) - \Pr(\mathbf{Hist}(\vec{X}) = t_{z+1} | \vec{X}_i = x_k) \Big) \\
=&\Pr(\mathbf{Hist}(\vec{X}) = t_q | \vec{X}_i = x_j) - \Pr(\mathbf{Hist}(\vec{X}) = t_0 | \vec{X}_i = x_k) \\
& \qquad \text{(Every term in the summation of differences cancels out.)} \\
=&\Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}) | \vec{X}_i = x_j) - \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}) | \vec{X}_i = x_k)
\end{aligned}$$

□

Remark. In this subsection's example, no matter the choice of $\mathcal{S}$, the set $\{t\colon f(t) \in \mathcal{S}\}$ forms one single trail, but this does not hold in general. Instead, to prove our main theorem we will partition this set into multiple trails, and apply Lemma 3 to simplify probabilities over each trail.

3.3.2 A simple application of trails technique: α-Biased Majority Rule

We prove the eDDP of α-biased majority voting rules f^v_α. The following theorem states that if the probability distribution of votes is split α-(1-α)), then $\delta = \Theta(\sqrt{1/n})$. As we will see in Theorem 3, this is the privacy achieved by the histogram function $\mathbf{Hist}$. By immunity to post-processing (Lem. 2), this implies that such probability distribution incurs the "worst case" privacy for a voting rule, which intuitively is due to the higher likelihood of ties (where changing one vote can change the winning candidate).

Theorem 2 (Exact DDP for α-biased majority rules)**.** *Fix two candidates $\{a, b\}$ and $\Delta \subseteq \Pi(\{a, b\})$ with $|\Delta| < \infty$. For any $\alpha \in (0, 1)$, the α-biased majority rule f^v_α is $(0, \delta, \Delta)$-eDDP for all n, where*

$$\delta = \max_{p = \pi(a)\colon \pi \in \Delta} \Theta\left(\sqrt{\frac{1}{n}} \left[\left(\frac{p}{\alpha}\right)^{\alpha} \left(\frac{1-p}{1-\alpha}\right)^{1-\alpha} \right]^n \right).$$

In particular, $\delta = \Theta\left(\sqrt{1/n}\right)$ if there exists $\pi \in \Delta$ with $\pi(a) = \alpha$; otherwise $\delta = \exp(-\Omega(n))$.

Proof. For any $\pi \in \Delta$, let $p = \pi(a)$. Let trails $\mathsf{T}_a = \{t : t = (k, n-k),\, k \ge \alpha n\}$ and $\mathsf{T}_b = \{t : t = (k, n-k),\, k < \alpha n\}$.

It follows that any histogram in T_a results in a being the winner, and any in T_b results in b as the winner. Also, Equation (3.7) implies we should *not* consider $\mathcal{S} = \{a, b\}$ nor $\mathcal{S} = \emptyset$ as otherwise that only gives us $\delta = 0$ (the default lower bound on δ). Thus, we only consider $\mathcal{S} = \{a\}$ (when winner is a, corresponding to trail T_a) or $\mathcal{S} = \{b\}$ (trail T_b). Then Equation (3.7) becomes (we disregard the value of i since votes are i.i.d.):

$$\delta = \max_{j\in\{a,b\},x,x'} \left[\Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathsf{T}_j | \vec{X}_i = x) - \Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathsf{T}_j | \vec{X}_i = x')\right] \quad \text{(Equation (3.7))}$$

$$= \max_{j\in\{a,b\},x,x'} \left[\Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}_j) | \vec{X}_i = x) - \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}_j) | \vec{X}_i = x')\right]. \quad \text{(Lemma 3)}$$

We first discuss $\mathcal{S} = \{a\}$ whose corresponding trail T_a starts at $\mathbf{Enter}(\mathsf{T}_a) = (n, 0)$ and exits at $\mathbf{Exit}(\mathsf{T}_a) = (\lceil \alpha n \rceil, \lfloor (1-\alpha)n \rfloor)$. Here, $x = a$ and $x' = b$ maximize δ. Then,

$$\Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}_a) | \vec{X}_i = b) \;=\; \Pr(\mathbf{Hist}(\vec{X}) = (n, 0) | \vec{X}_i = b) \;=\; 0,$$

and

$$\begin{aligned}
&\Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}_a) = \mathbf{End}(a) | \vec{X}_i = a) \\
&= \Pr(\mathbf{Hist}(\vec{X}) = (\lceil \alpha n \rceil, \lfloor (1-\alpha)n \rfloor) | \vec{X}_i = a) \\
&= \Pr(\mathbf{Hist}(\vec{X}) = (\lceil \alpha n \rceil - 1, \lfloor (1-\alpha)n \rfloor)) \\
&= p^{\lceil \alpha n \rceil - 1}(1-p)^{\lfloor (1-\alpha)n \rfloor} \frac{(n-1)!}{\lceil \alpha n - 1 \rceil ! \cdot \lfloor (1-\alpha)n \rfloor !} \\
&= \Theta\left[\frac{1}{\sqrt{n}} \cdot \left(\frac{pn}{\lceil \alpha n - 1 \rceil}\right)^{\lceil \alpha n - 1 \rceil} \cdot \left(\frac{(1-p)n}{\lfloor (1-\alpha)n \rfloor}\right)^{\lfloor (1-\alpha)n \rfloor}\right] \quad \text{(Stirling's formula)} \\
&= \Theta\left(\sqrt{\frac{1}{n}}\left[\left(\frac{p}{\alpha}\right)^{\alpha}\left(\frac{1-p}{1-\alpha}\right)^{1-\alpha}\right]^n\right).
\end{aligned}$$

The case for $\mathcal{S} = \{b\}$ is the same and Theorem 2 follows by maximizing δ over $\pi \in \Delta$.

□

3.4 The exact privacy of voting rules: general case

In this section, we apply the trails technique introduced in our previous section to the histogram function $\mathbf{Hist}$ and a class of voting rules called *generalized scoring rules (GSR)* (Definition 7), both for arbitrary number of candidates. We first study the $(\epsilon = 0, \delta, \Delta)$-exact DDP of the histogram function $\mathbf{Hist}$ (Theorem 3) in Section 3.4.1. Then, we characterise the $(\epsilon = 0, \delta, \Delta)$-exact DDP of GSRs (Theorem 4) in Section 3.4.2. As corollary, this characterization holds for most commonly-used voting rules (Corollary 2).

3.4.1 The exact DDP of the histogram function

As a complementary result to our Theorem 1 on the DDP of histograms, we present the exact DDP of $\mathbf{Hist}$ with $\epsilon = 0$. We show that the histogram function has eDDP of $\delta = \Theta\left(\frac{1}{\sqrt{n}}\right)$. As

we see later in the section about the eDDP of GSRs, this is exactly the "worst case" privacy for any GSR. This is intuitive: the privacy outputting the winner of an election should be at least as good as outputting the histogram of votes (which usually contains more information).

Theorem 3 (Exact DDP of histogram)**.** *Fix $\ell \geq 2$, $\mathcal{U} = \{x_1, \cdots, x_\ell\}$, and $\Delta \subseteq \Pi(\mathcal{U})$. If $\forall i \forall \pi \in \Delta : \pi[x_i] > 0$, then for all $n \in \mathbb{N}$, $\mathbf{Hist}$ of n voters is $\left(0, \delta(n) = \Theta\left(\sqrt{\frac{1}{n}}\right), \Delta\right)$-eDDP. Otherwise $\mathbf{Hist}$ is $(0, 1, \Delta)$-eDDP.*

We begin with a proof sketch then proceed with the full proof.

Proof Sketch of Theorem 3. By definition of (e)DDP, if there is some vote that has zero probability of being cast (which must the case if $\pi(x_i) = 0$ for some $\pi \in \Delta$ and x_i), then it must be that $\delta = 1$ as $\mathbf{Hist}$ is a deterministic function. This is by the same reasoning as to why we elect to use DDP instead of DP to analyze deterministic voting rules' privacy. To sketch a more intuitive version of the proof, we first present the case for $\ell = 2$.

Lemma 4 (Exact DDP for histogram, when $\ell = 2$)**.** *Fix $\mathcal{U} = \{x_1, x_2\}$ and $\Delta \subseteq \Pi(\mathcal{U})$. The histogram for n voters is $(0, \Theta(1/\sqrt{n}), \Delta)$-eDDP.*

Proof. Consider some $\pi \in \Delta$, and let $p = \pi(a)$. Without loss of generality (WLOG), let x and x' of Equation (3.6) be x_1 and x_2 respectively (otherwise, rename them). Then, the maximizing set $\mathcal{S}$ in Equation (3.6) is exactly the set of histograms such that

$$\Pr_{\vec{X} \in \pi}\left(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_1\right) > \Pr(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_2)$$

Since votes are i.i.d., these follow the binomial distribution (with n trials). Below we find that $\mathcal{S}$ is the set of histograms $(k, n-k)$ where $k > pn$.

$$\begin{aligned} &\Pr(\mathbf{Hist}(\vec{X}) = (k, n-k) | \vec{X}_i = x_1) > \Pr(\mathbf{Hist}(\vec{X}) = (k, n-k) | \vec{X}_i = x_2) \\ \implies &p^{k-1}(1-p)^{n-k}\frac{(n-1)!}{(n-k)!(k-1)!} > p^k(1-p)^{n-k-1}\frac{(n-1)!}{(n-k-1)!k!} \\ \implies &k > pn \end{aligned}$$

Thus, $\mathcal{S} = \{t = (k, n-k) \colon k > pn\}$. This set forms a trail T which starts from $\mathbf{Enter}(\mathsf{T}) = (n, 0)$ and exits at $\mathbf{Exit}(\mathsf{T}) = (pn+1, n-(pn+1))$. Thus,

$$\begin{aligned} \delta &= \Pr(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_1) - \Pr(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_2) && \text{(Equation (3.6))} \\ &= \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}) | \vec{X}_i = x_1) - \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}) | \vec{X}_i = x_2) && \text{(Lemma 3)} \\ &= \Pr(\mathbf{Hist}(\vec{X}) = (pn+1, n-(pn+1)) | \vec{X}_i = x_1) - \Pr(\mathbf{Hist}(\vec{X}) = (n, 0) | \vec{X}_i = x_2) \\ &= p^{pn}(1-p)^{n-pn-1}\frac{(n-1)!}{(pn)!(n-pn-1)!} \\ &\qquad \text{(When one vote is fixed to } x_2\text{, the probability of histogram being } (n,0) \text{ is zero.)} \\ &= \Theta(1/\sqrt{n}) && \text{(By applying Stirling's formula)} \end{aligned}$$

□

The general case of $\ell > 2$ can be extended from the above proof for $\ell = 2$, by also using the trails technique. In this case, we again assume WLOG that $x = x_1$ and $x' = x_2$. Let $t = (t_1, \cdots, t_\ell)$ be the histogram, where t_i counts the number of occurrences of x_i. We observe that, when votes are i.i.d, $t_3, \cdots, t_\ell$ are independent of t_1, t_2 when conditioned on the sum $s = t_1 + t_2$. This means that we can compute δ for general ℓ, as a sum

$$\delta = \sum_{0<s\leq n} \delta_s \Pr(\mathbf{Bin}(n, \pi(x_1) + \pi(x_2)) = s)$$

Where δ_s is the δ-value for $\ell = 2$, when there are s votes. Using Chernoff bound we see that $\mathbf{Bin}(n, \pi(x_1) + \pi(x_2))$ is concentrated at its mean $n(\pi(x_1) + \pi(x_2))$. Plugging in the result for $\ell = 2$, we get $\delta = \Theta\left(\frac{1}{\sqrt{n(\pi(x_1)+\pi(x_2))}}\right)$. This is $\Theta(1/\sqrt{n})$ when $\pi(x_1) + \pi(x_2) > 0$ (recall these probabilities are constants). □

3.4.1.1 Full proof of Theorem 3

Below we present the full proof of Theorem 3, using Lemma 4 which showed the case for $\ell = 2$.

Let $p_{\min} = \min_{\pi\in\Delta; i\neq j\in[\ell]}(\pi(x_i) + \pi(x_j))$. By definition of (e)DDP, if there is an x_i, $\pi \in \Delta$ such that $\pi(x) = 0$ then only $\delta = 1$ satisfies the inequality in the definition of (e)DDP. Thus, below we will assume that for all x, $\pi(x) > 0$, and thus $p_{\min} > 0$.

Proof of Theorem 3, Exact DDP of Histogram. Consider any $\pi \in \Delta$, and let $p_i = \pi(x_i)$. Like in the $\ell = 2$ case, without loss of generality, we can let $x = x_1$ and $x' = x_2$ (otherwise, rename them). Then, the maximizing set $\mathcal{S}$ (similar to when $\ell = 2$) is exactly the set of histograms such that

$$\Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_1) > \Pr_{\vec{X}\sim\pi}(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_2)$$

(We will implicitly assume $\vec{X} \sim \pi$ from now on) Since we have i.i.d. votes, the histogram follows the multinomial distribution (with n trials). For any $0 < s \leq n$, $(t_3, \cdots, t_\ell)$ where $t_3 + \cdots + t_\ell = n - s$, and $k \leq s$:

$$\Pr(\mathbf{Hist}(\vec{X}) = (k, s-k, t_3, \cdots, t_\ell) | \vec{X}_i = x_1) > \Pr(\mathbf{Hist}(\vec{X}) = (k, s-k, t_3, \cdots, t_\ell) | \vec{X}_i = x_2)$$

$$p_1^{k-1} p_2^{n-k} p_3^{t_3} \cdots p_\ell^{t_\ell} \frac{(n-1)!}{(k-1)!(s-k)!t_3! \cdots t_\ell!} > p_1^k p_2^{n-k-1} p_3^{t_3} \cdots p_\ell^{t_\ell} \frac{(n-1)!}{(s-k-1)!k!t_3! \cdots t_\ell!}$$

$$\frac{p_2}{s-k} > \frac{p_1}{k}$$

$$k > \left(\frac{p_1}{p_1+p_2}\right) s$$

Thus, the set $\mathcal{S} = \left\{t = (k, s-k, t_3, \cdots, t_\ell) \colon k > \left(\frac{p_1}{p_1+p_2}\right) s\right\}$.

Let $p = \frac{p_1}{p_1+p_2}$. For each $0 < s \le n$ and $(t_3, \cdots, t_\ell)$ which sum to $n-s$ (i.e. $t_3 + \cdots + t_\ell = n-s$), let $\mathsf{T}_{s,(t_3,\cdots,t_\ell)}$ be the trail starting from $\mathbf{Enter}(\mathsf{T}_{s,(t_3,\cdots,t_\ell)}) = (s, 0, t_3, \cdots, t_\ell)$ and exiting at $\mathbf{Exit}(\mathsf{T}_{s,(t_3,\cdots,t_\ell)}) = (ps+1, s-(ps+1), t_3, \cdots, t_\ell)$. The set $\mathcal{S}$ then can be partitioned into such trails. Thus,

$$
\begin{aligned}
\delta &= \Pr(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_1) - \Pr(\mathbf{Hist}(\vec{X}) \in \mathcal{S} | \vec{X}_i = x_2) \\
&= \sum_{\mathsf{T}_{s,(t_3,\cdots,t_\ell)}} \Pr(\mathbf{Hist}(\vec{X}) \in \mathsf{T}_{s,(t_3,\cdots,t_\ell)} | \vec{X}_i = x_1) - \Pr(\mathbf{Hist}(\vec{X}) \in \mathsf{T}_{s,(t_3,\cdots,t_\ell)} | \vec{X}_i = x_2) \\
&= \sum_{\mathsf{T}_{s,(t_3,\cdots,t_\ell)}} \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Exit}(\mathsf{T}_{s,(t_3,\cdots,t_\ell)}) | \vec{X}_i = x_1) \\
&\qquad - \Pr(\mathbf{Hist}(\vec{X}) = \mathbf{Enter}(\mathsf{T}_{s,(t_3,\cdots,t_\ell)}) | \vec{X}_i = x_2) \qquad \text{(By Lemma 3)} \\
&= \sum_{0<s\le n} \sum_{\substack{(t_3,\cdots,t_\ell) \\ t_3+\cdots+t_\ell = n-s}} \Pr(\mathbf{Hist}(\vec{X}) = (ps+1, s-(ps+1), t_3, \cdots, t_\ell) | \vec{X}_i = x_1) \\
&\qquad\qquad - \Pr(\mathbf{Hist}(\vec{X}) = (s, 0, t_3, \cdots, t_\ell) | \vec{X}_i = x_2)
\end{aligned}
$$

Now let us consider these two probabilities. Consider the distribution $\vec{X}_{-i}$, which is $\vec{X}$ but without the ith row. Let the random variables of the individual components of $\mathbf{Hist}(\vec{X}_{-1})$ be $(a_1, \cdots, a_\ell)$. Since votes are i.i.d., for any $(t_1, \cdots, t_\ell)$,

$$
\begin{aligned}
&\Pr(\mathbf{Hist}(\vec{X}) = (t_1, \cdots, t_\ell) | \vec{X}_i = x_1) \\
&= \Pr(\mathbf{Hist}(\vec{X}_{-i}) = (t_1 - 1, t_2, t_3, \cdots, t_\ell)) \\
&= \Pr((a_1, \cdots, a_\ell) = (t_1 - 1, t_2, t_3, \cdots, t_\ell)) \ \text{(Recall these } a\text{'s are components of } \mathbf{Hist}(\vec{X}_{-i})) \\
&= \Pr((a_1, \cdots, a_\ell) = (t_1 - 1, t_2, t_3, \cdots, t_\ell) | a_1 + a_2 = s) \times \Pr(a_1 + a_2 = s) \\
&= \Pr((a_1, a_2) = (t_1 - 1, t_2) \, | a_1 + a_2 = s) \\
&\qquad \times \Pr((a_3, \cdots, a_\ell) = (t_3, \cdots, t_\ell) \, | a_1 + a_2 = s) \times \Pr(a_1 + a_2 = s)
\end{aligned}
$$

(By Lemma 5 proven below, (a_1, a_2) and $(a_3, \cdots, a_\ell)$ are independent conditioned on $a_1 + a_2 = s$)

Similar to the $\ell = 2$ case, $\Pr(\mathbf{Hist}(\vec{X}) = (s, 0, t_3, \cdots, t_\ell) | \vec{X}_i = x_2) = 0$. This is because when one vote is fixed to x_2, it is impossible to have zero in the second component in the histogram (which is the number of occurrences of x_2). Thus,

$$\begin{aligned}\delta &= \sum_{0<s\le n} \sum_{\substack{(t_3,\cdots,t_\ell)\\ t_3+\cdots+t_\ell=n-s}} \Pr((a_1,a_2)=(ps,s-(ps+1))|a_1+a_2=s) \\ &\qquad \times \Pr((a_3,\cdots,a_\ell)=(t_3,\cdots,t_\ell)|a_1+a_2=s)\times \Pr(a_1+a_2=s) \\ &= \sum_{0<s\le n} \Pr((a_1,a_2)=(ps,s-(ps+1))|a_1+a_2=s)\times \Pr(a_1+a_2=s) \\ &\qquad \times \sum_{\substack{(t_3,\cdots,t_\ell)\\ t_3+\cdots+t_\ell=n-s}} \Pr((a_3,\cdots,a_\ell)=(t_3,\cdots,t_\ell)|a_1+a_2=s)\end{aligned}$$

(Factor out the common terms $\Pr((a_1,a_2)=(ps,s-(ps+1))|a_1+a_2=s)$ and $\Pr(a_1+a_2=s)$)

$$= \sum_{0<s\le n} \Pr((a_1,a_2)=(ps,s-(ps+1))|a_1+a_2=s)\times \Pr(a_1+a_2=s)$$

(For any s, the second sum equals one.)

Where $\Pr((a_1,a_2)=(ps,s-(ps+1))|a_1+a_2=s)$ is the δ value for histogram when $\ell=2$, the vote distribution is $\pi'\in\Pi(\{x_1,x_2\})$, where $\pi'(x_1)=\frac{p_1}{p_1+p_2}$, and number of voters is s (we refer to Lemma 4 of the $\ell=2$ case for this claim). We denote this δ by δ_s. Moreover,

$$\Pr(a_1+a_2=s)=\Pr(\mathbf{Bin}(n,p_1+p_2)=s)$$

We denote $p'=p_1+p_2$. Then, $\mathbf{Bin}(n,p')$ is the binomial distribution with n trials and probability $p'=p_1+p_2$ (recall that $p_i=\pi(x_i)$). Then

$$\begin{aligned}\delta &= \sum_{0<s\le n} \delta_s \Pr\left(\mathbf{Bin}\left(n,p'\right)=s\right) \\ &= \sum_{\substack{s\ge\left(1-\sqrt{\frac{3}{4}}\right)np'\\ s\le\left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n,p'\right)=s\right)\times\delta_s + \sum_{\substack{s<\left(1-\sqrt{\frac{3}{4}}\right)np'\\ s>\left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n,p'\right)=s\right)\times\delta_s\end{aligned}$$

Lower bound of δ:

$$\begin{aligned}
\delta &\geq \sum_{\substack{s\geq\left(1-\sqrt{\frac{3}{4}}\right)np' \\ s\leq\left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n,p'\right)=s\right)\times\delta_s \\
&\geq \delta_{\left(1+\sqrt{\frac{3}{4}}\right)np'} \times \sum_{\substack{s\geq\left(1-\sqrt{\frac{3}{4}}\right)np' \\ s\leq\left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n,p'\right)=s\right)
\end{aligned}$$

(δ_s decreases with larger s (more votes implies more privacy), so $\delta_{\left(1+\sqrt{\frac{3}{4}}\right)np'}$ is the minimum.)

$$= \delta_{\left(1+\sqrt{\frac{3}{4}}\right)np'} \times \left[1-\Pr\left(\mathbf{Bin}(n,p')>\left(1+\sqrt{\frac{3}{4}}np'\right)\right)-\Pr\left(\mathbf{Bin}(n,p')<\left(1-\sqrt{\frac{3}{4}}np'\right)\right)\right]$$

By Chernoff bound for binomial distribution, for any $0<\beta<1$, we have:

$$\Pr\left(\mathbf{Bin}\left(n,p'\right)>\left(1+\beta\right)\mu\right)\leq e^{-\frac{\beta^2\mu}{3}}$$
$$\Pr\left(\mathbf{Bin}\left(n,p'\right)<\left(1-\beta\right)\mu\right)\leq e^{-\frac{\beta^2\mu}{2}}$$

Where $\mu=np'$ is the mean of $\mathbf{Bin}\left(n,np'\right)$. Now let $\beta=\sqrt{\frac{3}{4}}$, which is between 0 and 1. Then,

$$\begin{aligned}
1 &\geq \left[1-\Pr\left(\mathbf{Bin}\left(n,p'\right)>\left(1+\sqrt{\frac{3}{4}}\right)np'\right)-\Pr\left(\mathbf{Bin}\left(n,p'\right)<\left(1-\sqrt{\frac{3}{4}}\right)np'\right)\right] \\
&\geq 1-e^{-\frac{3}{4}\frac{\mu}{3}}-e^{-\frac{3}{4}\frac{\mu}{2}} \\
&= 1-e^{-\frac{np'}{2}}-e^{-\frac{3np'}{2}} \\
&\qquad\qquad \text{(For large enough } n,\ np'\geq 1, \text{ so } e^{-\frac{np'}{2}}\leq e^{-1/2} \text{ and } e^{-\frac{3np'}{2}}\leq e^{-3/2}) \\
&\geq 1-e^{-1/2}-e^{-3/2}\geq\frac{1}{10}
\end{aligned}$$

Which means $\left[1-\Pr\left(\mathbf{Bin}(n,p')>\left(1+\sqrt{\frac{3}{4}}\right)np'\right)-\Pr\left(\mathbf{Bin}(n,p')<\left(1-\sqrt{\frac{3}{4}}\right)np'\right)\right]=\Theta(1)$.

By Stirling formula, we have

$$\delta_{\left(1+\sqrt{\frac{3}{4}}\right)np'} = \Theta\left(\frac{1}{\sqrt{\left(1+\sqrt{\frac{3}{4}}\right)np'}}\right)$$
$$= \Theta\left(\frac{1}{\sqrt{np'}}\right)$$

(Recall we assumed the maximizing x, x' are x_1, x_2, up to renaming the x_i's, and that $p' = p_1 + p_2$)

$$= \Theta\left(\frac{1}{\sqrt{np_{\min}}}\right)$$ (In general, $p_{\min} = \min_{i \neq j \in [\ell]}(p_i + p_j)$.)

Which gives us the lower bound $\delta \geq \Theta\left(\frac{1}{\sqrt{np_{\min}}}\right)$.

Upper bound of δ:

$$\delta = \sum_{\substack{s \geq \left(1-\sqrt{\frac{3}{4}}\right)np' \\ s \leq \left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n, p'\right) = s\right) \times \delta_s$$
$$+ \sum_{\substack{s < \left(1-\sqrt{\frac{3}{4}}\right)np' \\ s > \left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n, p'\right) = s\right) \times \delta_s$$

Since $\delta_s \leq 1$ for all s and $\sum_{\substack{s \geq \left(1-\sqrt{\frac{3}{4}}\right)np' \\ s \leq \left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n, p'\right) = s\right) \leq 1$

$$\leq \max_{\left(1-\sqrt{\frac{3}{4}}\right)np' \leq s \leq \left(1+\sqrt{\frac{3}{4}}\right)np'} (\delta_s) + \sum_{\substack{s < \left(1-\sqrt{\frac{3}{4}}\right)np' \\ s > \left(1+\sqrt{\frac{3}{4}}\right)np'}} \Pr\left(\mathbf{Bin}\left(n, p'\right) = s\right)$$

$$= \delta_{\left(1-\sqrt{\frac{3}{4}}\right)np'} + \Pr\left(\mathbf{Bin}\left(n, p'\right) < \left(1-\sqrt{\frac{3}{4}}\right)np'\right) + \Pr\left(\mathbf{Bin}\left(n, p'\right) > \left(1+\sqrt{\frac{3}{4}}\right)np'\right)$$

$$\leq \delta_{\left(1-\sqrt{-\frac{3}{4}}\right)np'} + e^{-\frac{np'}{2}} + e^{\frac{3np'}{2}}$$ (By Chernoff bound for binomial)

$$\leq \delta_{\left(1-\sqrt{-\frac{3}{4}}\right)np'} + 2\sqrt{\frac{1}{np'}}$$ (Since $np' \geq 0$, both $e^{-\frac{np'}{2}}, e^{\frac{3np'}{2}} \leq \sqrt{\frac{1}{np'}}$)

By Stirling's formula, $\delta_{\left(1-\sqrt{-\frac{3}{4}}\right)np'} = \Theta\left(\frac{1}{\sqrt{(1-\sqrt{-\frac{3}{4}})np'}}\right)$

$$= \Theta\left(\sqrt{\frac{1}{np'}}\right)$$

As is with the lower bound, in general (without assuming $(x, x') = (x_1, x_2)$), we have $p' = p_{\min} = \min_{i \neq j \in [\ell]}(p_i + p_j)$. Since both lower and upper bounds of δ are $\Theta\left(\sqrt{\frac{1}{np_{\min}}}\right)$, $\delta = \Theta\left(\frac{1}{\sqrt{np_{\min}}}\right) = \Theta\left(\frac{1}{\sqrt{n}}\right)$ as $p_{\min}$ is a constant. □

We now present the lemma on conditional independence which is used in the proof of Theorem 3.

Lemma 5 (Conditional independence). *Let $\mathcal{U} = \{x_1, \cdots, x_\ell\}$ and $\pi \in \Delta(\mathcal{U})$. Let $\#x_i$ denote the r.v. of the number of occurrences of the vote x_i in π. Then, for all $0 \leq s \leq n$, the random variables $(\#x_1, \#x_2)$ and $(\#x_3, \cdots, \#x_\ell)$ are independent conditioned on $\#x_1 + \#x_2 = s$. In other words, for any $(t_1, \cdots, t_\ell)$ such that $\sum_i t_i = n$, we have*

$$\begin{aligned}&\Pr((\#x_1, \cdots, \#x_\ell) = (t_1, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s) \\ =&\Pr((\#x_1, \#x_2) = (t_1, t_2) \mid \#x_1 + \#x_2 = s) \times \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s)\end{aligned}$$

Proof. We equivalently show that

$$\begin{aligned}&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s) \\ =&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s \wedge (\#x_1, \#x_2) = (t_1, t_2))\end{aligned} \tag{3.9}$$

Now, conditioned on there being exactly s people who voted x_1 or x_2, let $D_1 > D_2 > \cdots > D_s$ denote the random variables of the indices of the votes in the profile which voted for x_1 or x_2, in ascending order. By total probability, the left hand side of Equation 3.9 is:

$$\begin{aligned}&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s) \\ =&\sum_{d_1 > d_2 > \cdots > d_s} \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s \wedge (D_1, \cdots, D_s) = (d_1, \cdots, d_s)) \\ &\qquad \times \Pr((D_1, \cdots, D_s) = (d_s, \cdots, d_s) \mid \#x_1 + \#x_2 = s)\end{aligned}$$

We already assume there are exactly s votes for x_1 or x_2, so

$$\begin{aligned}&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s) \\ =&\sum_{d_1 > d_2 > \cdots > d_s} \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid (D_1, \cdots, D_s) = (d_1, \cdots, d_s)) \\ &\qquad \times \Pr((D_1, \cdots, D_s) = (d_1, \cdots, d_s))\end{aligned}$$

The right hand side of Equation 3.9 is:

$$
\begin{aligned}
&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s \wedge (\#x_1, \#x_2) = (t_1, t_2)) \\
&= \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s \wedge (\#x_1, \#x_2) = (t_1, t_2)) \\
&= \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid (\#x_1, \#x_2) = (t_1, t_2)) && \text{(Since we assume } t_1 + t_2 = s\text{)} \\
&= \sum_{d_1 > d_2 > \cdots > d_s} \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid (\#x_1, \#x_2) = (t_1, t_2) \wedge (D_1, \cdots, D_s) = (d_1, \cdots, d_s)) \\
&\qquad \times \Pr((D_1, \cdots, D_s) = (d_1, \cdots, d_s) \mid (\#x_1, \#x_2) = (t_1, t_2)) && \text{(By total probability,)}
\end{aligned}
$$

Since each vote is independent, $(\#x_3, \cdots, \#x_\ell)$ is independent of $(\#x_1, \#x_2)$. Moreover, the vote indices $(D_1, \cdots, D_s)$ are independent of $(\#x_1, \#x_2)$. As votes are i.i.d., $(\#x_1, \#x_2)$ does not depend on the value of $(d_1, \cdots, d_s)$. Thus,

$$
\begin{aligned}
&\Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid \#x_1 + \#x_2 = s \wedge (\#x_1, \#x_2) = (t_1, t_2)) \\
&= \sum_{d_1 > d_2 > \cdots > d_s} \Pr((\#x_3, \cdots, \#x_\ell) = (t_3, \cdots, t_\ell) \mid (D_1, \cdots, D_s) = (d_1, \cdots, d_s)) \\
&\qquad \times \Pr((D_1, \cdots, D_s) = (d_1, \cdots, d_s))
\end{aligned}
$$

This concludes that the left hand side and right hand side probabilities of Equation 3.9 are equal. The random variables $(\#x_1, \#x_2)$ are independent conditioned on $(\#x_1, \#x_2)$.

□

3.4.2 The exact DDP of GSRs

In this section we characterize the privacy of a class of voting rules called *generalized scoring rules* (GSR). Recall that a voting rule f^v is a GSR if (1) the votes are rankings (linear orders) over the candidates, and (2) can be described using functions f^{gsr}, g^{gsr}. Roughly, the function f^{gsr} takes a vote and outputs a vector of numbers called the *score*, and g^{gsr} outputs the winning candidate from the sum of the score vectors of all votes. Most popular voting rules (i.e., Borda, Plurality, k-approval and ranked pairs) are GSRs. See Example 1 and Example 2 in the preliminaries (Ch. 2) for f^{gsr}, g^{gsr} for plurality rule and majority rule.

We now define a set of properties of GSR voting rules to present our characterization of eDDP in Theorem 4: *canceling-out*, *monotonicity*, and *local stability*.

Remark about $f^v(\pi)$ notation: In the following section, we will slightly abuse notation by also defining voting rules f^v on *weighted* profiles (where votes are weighted by a real number) denoted with lower case vector notation e.g., $\vec{p}$. This is possible for all GSRs due to the linearity of f^{gsr}. We also consider $f^v(\pi)$ when π is a distribution, which denotes the output of f^v when the voters cast fractional votes according π.

Definition 20 (Canceling-out, Monotonicity, and Local stability)**.** A voting rule f^v satisfies *canceling-out* if for any profile $\vec{X}$, adding a copy of every ranking does not change the winner. That is, $f^v(\vec{X}) = f^v(\vec{X} \cup \mathcal{L}(C))$.

A voting rule satisfies *monotonicity* one cannot prevent a candidate from winning by raising its ranking in a vote while maintaining the order of other candidates.

A voting rule f^v satisfies *local stability* if there exist locally stable profile. A profile $\vec{X}^*$ is *locally stable* (to f^v), if there exists a candidate a, a vote/ranking W, and another vote/ranking V that is obtained from W by raising the position of a without changing the order of other candidates, such that for any $\vec{X}'$ in the γ neighbourhood of $\vec{X}^*$ in terms of L_∞ norm, we have (1) $f^v(\vec{X}') \neq a$, and (2) the winner is a when all votes W in $\vec{X}'$ are replaced by V.

We also define what it means for a distribution to be *unstable* for a GSR f^v.

Definition 21 (Unstable distributions)**.** Given a GSR f^v, a distribution π over $\mathcal{U}$ is *unstable*, if for any $\epsilon > 0$, there exists $\vec{p}$ and $\vec{q}$ with $\|\vec{p}\|_2 = \|\vec{q}\|_2 < \epsilon$, such that $f^v(\pi + \vec{q}) \neq f^v(\pi + \vec{p})$ (i.e., the winners are not the same if the fractional profile is slightly perturbed), where $\|\cdot\|_2$ is the ℓ_2-norm.

Theorem 4 (Dichotomy of exact DDP for GSR)**.** *Fix $m \geq 2$ and $\Delta \subseteq \Pi(\mathcal{L}(\mathcal{C}))$ with $|\Delta| < \infty$. For any n, any GSR f^v that satisfies monotonicity, local stability, and canceling-out is $(0, \delta, \Delta)$-DDP, where δ is*

- $\Theta(\sqrt{1/n})$*, if Δ contains the uniform distribution over $\mathcal{L}(\mathcal{C})$, or*
- $\exp(-\Omega(n))$*, if Δ does not contain any unstable distribution.*

We first sketch the proof below.

Proof sketch for Theorem 4. We first prove the $\delta = \exp[-\Omega(n)]$ case. Recalling Sec. 3.3.1 and the proof of Theorem 2, we know that δ is closely related to the probability of $\mathbf{End}(a)$ for some $a \in \mathcal{C}$, where intuitively, $\mathbf{End}(a)$ is the set of profiles that, if some votes are changed, a is no longer the winner. It turns out that this is also the case for any GSR f^v that also satisfies monotonicity. For a preference profile P, let $P - V$ be shorthand for "P minus a vote V" (if P has a vote V), and $P + W$ shorthand for "P with an additional vote W". Applying our trails technique, we have

$$\delta \leq \max_a \sum_{P \in \mathbf{End}(a)} \Pr(P - V),$$

where V is a vote s.t. there exists vote W with $f^v(P - V + W) \neq a$. Thus, we know δ is upper bounded by the probability of all profiles $(P - V)$ being "close" to a tie of voting rule r. For any unstable distribution π, we can prove that the center of π is reasonably "far" away from any profile in $\mathbf{End}(a)$ (or "far" away from any ties). Then, the exponential upper bound follows after Chernoff bound and union bound. The proof for this part is similar to the analysis of probabilities of tied profiles as in [127].

We now move on to the $\delta = \Theta(\sqrt{1/n})$ case. The upper bound $O(\sqrt{1/n})$ also derived from the trails technique's result: $\delta \leq \max_a \sum_{P \in \mathbf{End}(a)} \Pr(P - V)$. The general framework of

our proof is similar with the $\delta = \exp[-\Omega(n)]$ case. Since adding any vote to a uniform profile results in a new winner, we know the uniform distribution of preferences is always an unstable distribution when requirements in Theorem 4 are met. Thus, we can prove that the center of the profiles' distribution (multinomial distribution in $m!$-dimensional space) is "close" to a tie. Then, we apply Stirling's formula to each trails and give an upper bounds to $\Pr(P - V)$ for profiles $P \in \mathbf{End}(a)$.

For the lower bound $\Omega(\sqrt{1/n})$, canceling-out and local stability properties are used to construct a "good" subset of profiles. At a high level, canceling-out ensures that the constructed subset is large enough, and locally stability ensures the trails constructed from the selected subset is long enough. Our subset is contracted by certain profiles with $O(\sqrt{n})$ distance[2] from the center of profile distribution in the direction of local stable profile. Giving a lower bound to the $\Pr(P - V)$ for any profile P in our selected subset is the most non-trivial part of this proof and is quite different from the proof in [127]. Unlike the profiles P in our selected subset of profiles, $P - V$ do not necessarily concentrated in a specific region in the space of profiles. Here, we use a non-i.i.d. version of Lindeberg-Levy central limit theorem [78] to analyze the multinomial distribution of $m!$ kinds of votes.

□

We provide the full proof for Theorem 4 below.

3.4.2.1 General notation used in the proof

Throughout the proof in Theorem 4, we will use $\vec{\pi}$ to denote the probability distribution in each vote, given the (i.i.d.) profile distribution D, and $\pi[j]$ denote the probability of j-th kind of ranking. We denote by $P[V]$ as the number of V votes in P and $P[j]$ the number of j-th type of vote in P. For any $P \in \mathcal{R}_n$, let $\mathbf{Piv}(P) = \mathbf{End}(a) \cap \mathsf{T}_{P,V,W,\infty}$ denote the intersection of $\mathbf{End}(a)$ and the V-W trail starting at P. That is, $\mathbf{Piv}(P) = P + l(V - W)$ for some $l \in \mathbb{Z}$, $r(\mathbf{Piv}(P)) = a$, and $r(\mathbf{Piv}(P) - V + W) \neq a$. Recall $P + l(V - W)$ means adding l votes of type V to, and removing l votes of type W from profile P.

3.4.2.2 Full proof for Theorem 4

Proof of Theorem 4, (Exact) DDP for GSR. To present the result, we first introduce an equivalent definition of GSR that is similar to the ones used in [Xia and Conitzer, 2009; Mossel *et al.*, 2013].

Definition 22 (The (H, g_H) definition of GSR)**.** A GSR over m candidates is defined by a set of hyperplanes $H = \{\vec{h}_1, \ldots, \vec{h}_R\} \subseteq \mathbb{R}^{m!}$ and a function $g_H : \{+, 0, -\}^{|H|} \to C$. For any anonymous profile (recall GSRs are anonymous voting rules) $\vec{p} \in \mathbb{R}^{m!}$, we let $H(\vec{p}) = (\mathbf{Sign}(\vec{h}_1 \cdot \vec{p}), \ldots, \mathbf{Sign}(\vec{h}_R \cdot \vec{p}))$, where $\mathbf{Sign}(x)$ is the sign ($+, -$ or 0) of a number x. We let the winner be $g_H(H(\vec{p}))$.

[2] we use ℓ_2 distance in the $m!$-dimensional space of profile.

That is, to determine the winner, we first use each hyperplane in H to classify the profile $\vec{p}$, to decide whether $\vec{p}$ is on the positive side $(+)$, negative side $(-)$, or is contained in the hyperplane (0). Then g_H is used to choose the winner from $H(\vec{p})$. We refer to this definition the (H, g_H) definition. In the next claim, we show the equivalence of two definitions of GSR.

Claim 1. *The (H, g_H) definition of GSR is equivalent to the (f^{gsr}, g^{gsr}) definition of GSR in Definition 7.*

Proof for Claim 1. We first show that any (H, g_H) GSR can be represented by a (f^{gsr}, g^{gsr}) GSR in the following way: for each ranking V, we let $f^{gsr}(V) = (\vec{h}_1 \cdot \vec{e}_V, h_2 \cdot \vec{e}_V, \ldots, \vec{h}_R \cdot \vec{e}_V, 0)$. Then, the g^{gsr} function mimics g_H by only focusing on orderings between the kth component of $f^{gsr}(P)$ and the last component, which is always 0, for all $k \leq R$. More precisely, ordering between the kth component of $f^{gsr}(P)$ and 0 uniquely determines $\mathbf{Sign}(\vec{h}_k \cdot \vec{p})$.

We now prove that any (f^{gsr}, g^{gsr}) GSR can be represented by an (H, g_H) GSR. For any pair of distinct component $k_1, k_2 \leq K$, we introduce a hyperplane $\vec{h}_{k_1,k_2} = ([f^{gsr}(V)]_{k_1} - [f^{gsr}(V)]_{k_2})_{V \in \mathcal{L}(C)}$. Therefore, for any profile $\vec{p}$, $\vec{h}_{k_1,k_2} \cdot \vec{p} = [f^{gsr}(\vec{p})]_{k_1} - [f^{gsr}(\vec{p})]_{k_2}$. The sign of $\vec{h}_{k_1,k_2} \cdot \vec{p}$ corresponds to the order between $[f^{gsr}(\vec{p})]_{k_1}$ and $[f^{gsr}(\vec{p})]_{k_2}$. Then, g_H mimics g^{gsr}.

(End of Claim 1 proof)

□

For the remainder of the proof we will mainly work with the hyperplanes definition. Below, we define some general notation that will be used for the proof.

We will characterize eDDP under uniform distribution and give an exponential upper bound on DDP under some other distributions. For any pair of $\vec{\pi}$ and $\vec{h}$, we let $\mathbf{Dist}(\vec{\pi}, \vec{h}) = \frac{\vec{\pi} \cdot \vec{h}}{||\vec{h}||_2}$ to denote the distance between hyperplane $\vec{h} \cdot \vec{p} = 0$ and vector $\vec{\pi}$.

We first show that w.l.o.g. we can assume that all hyperplanes in H passes $\vec{1}$.

Lemma 6. *A GSR satisfies canceling-out, if and only if there exists another equivalent GSR $r = (H, g_H)$, where all hyperplanes pass $\vec{1}$.*

Proof. The "if" direction is straightforward: If hyperplanes in H pass $\vec{1}$, i.e. $\vec{h} \cdot \vec{1} = 0$, then adding one vote of each type (i.e., adding a vector $\vec{1}$ to the profile) does not change the winner—which is exactly the definition of canceling-out. To prove the "only if" part, it suffices to prove that g_H does not depend on outcomes of hyperplanes in H that does not pass $\vec{1}$, i.e., $\vec{h} \cdot \vec{1} \neq 0$. W.l.o.g. let $\vec{h}_1 \in H$ denote the hyperplane that does not pass $\vec{1}$. We will prove that for any $\vec{u}_{-1} \in \{-1, 0, 1\}^{L-1}$ and any $u_1, u_1' \in \{-1, 0, 1\}$, such that there exist profiles P, Q with $H(P) = (u_1, \vec{u}_{-1})$ and $H(Q) = (u_1', \vec{u}_{-1})$, we have $g_H(u_1, \vec{u}_{-1}) = g_H(u_1', \vec{u}_{-1})$.

For the sake of contradiction, suppose this does not hold and let P, Q be the profiles such that $H(P)$ and $H(Q)$ differ on the first coordinate, and $r(P) \neq r(Q)$. Then, for sufficiently large n we have that $H(P + n\mathcal{L}(\mathcal{C})) = H(Q + n\mathcal{L}(\mathcal{C}))$. This is because for any $\vec{h} \in H$ that passes $\vec{1}$, we have $\vec{h} \cdot (P + n\mathcal{L}(\mathcal{C})) = \vec{h} \cdot P = \vec{h} \cdot (Q + n\mathcal{L}(\mathcal{C}))$. For any $\vec{h} \in H$ that does not pass $\vec{1}$, we have $\vec{h} \cdot (P + n\mathcal{L}(\mathcal{C})) = \vec{h} \cdot P + n\vec{h} \cdot 1$, and when n is sufficiently large, the sign of

$\vec{h} \cdot (P + n\mathcal{L}(\mathcal{C}))$ is the same as the sign of $n\vec{h} \cdot 1$, which is the sign of $\vec{h} \cdot (Q + n\mathcal{L}(\mathcal{C}))$. This means that $\mathbf{Sign}(\vec{h} \cdot P) = \mathbf{Sign}(\vec{h} \cdot (P + n\mathcal{L}(\mathcal{C}))) = \mathbf{Sign}(\vec{h} \cdot (Q + n\mathcal{L}(\mathcal{C}))) = \mathbf{Sign}(\vec{h} \cdot Q)$, which is a contradiction.

(End of Lemma 6 proof)

□

Let f^v be a GSR (which can also be equivalently described by the hyperplanes definition as $r = (H, g_H)$), P^* be the locally stable profile and a be the candidate, V, W be the rankings as in the statement of Definition 20. W.l.o.g. suppose V is the first type ranking and W is the second type ranking. In other words, V (respectively, W) is the first (respectively, second) coordinate in the n-profiles space. We will show that the exact DDP bound is achieved when $\mathcal{S}$ is the set of all profiles where the winner is a.

We recall that for any profile P, a pair of different votes V, W. and a length $q \in \mathbb{N}$, $\mathsf{T}_{P,V,W,q}$ is the trail starting at P, going along the $V - W$ direction, and contains q profiles. We let $\mathsf{T}_{P,V,W,\infty} = \max_q \mathsf{T}_{P,V,W,q}$ denote the longest $V - W$ trail starting at P. For a GSR f^v, we define $\mathbf{End}(a)$ below. As we see, there are no W votes in $\mathbf{End}(a)$.

Definition 23 ($\mathbf{End}(\cdot)$)**.** Let f^v be a GSR and a be a candidate. We define

$$\mathbf{End}(a) = \{\mathbf{Exit}(\mathsf{T}_{P,V,W,\infty}) : \forall\, V, W \in \mathcal{U},\ f^v(P) = a\}$$

.

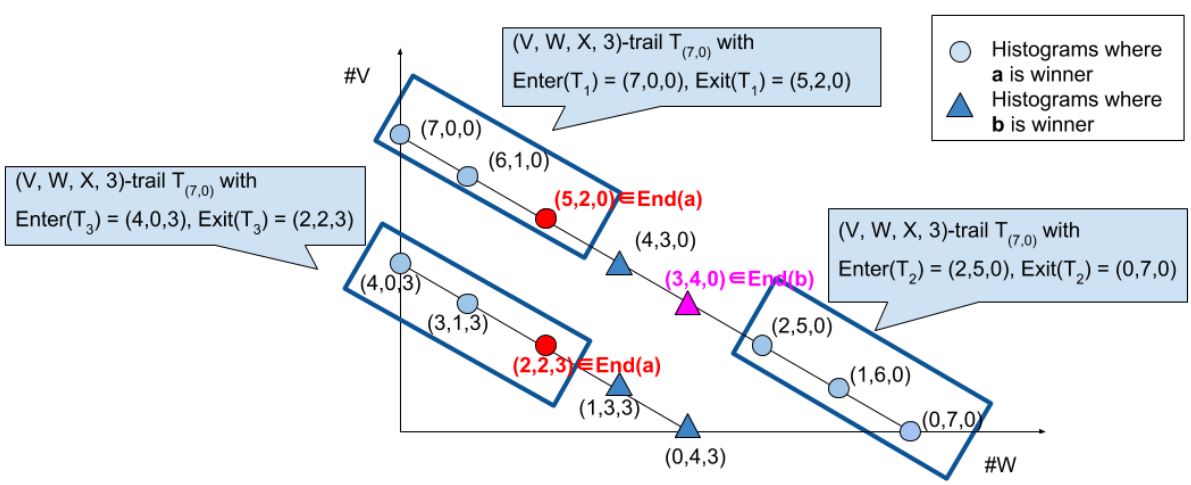

Figure 3.2: Example of $\mathbf{End}(a)$ and $\mathbf{End}(b)$, for 3-candidate case. The 3 kinds of votes other than V, W and X are not shown to simplify notations. Number of unshown votes are considered as constant.

Because f^v satisfies monotonicity, for any profile P such that $f^v(P) = a$, we must have that a is the winner under all profiles in the V-W trail starting at P. Therefore, $\mathcal{S}$ can be partitioned

into multiple non-overlapping trails, each of which starts at a different profile, where a is the winner, and a is no longer the winner if we go one step into the W-V direction. Formally, we let $\mathbf{End}(a)$ (shown in Figure 3.2) denote all n-profiles P such that (1) $f^v(P) = a$ and (2) $f^v(P + W - V) \neq a$. Then, we define a partition $\mathcal{S}_a$ as follows.

$$\mathcal{S}_a = \{P : f^v(P) = a\} = \bigcup_{P \in \mathbf{End}(a)} \mathsf{T}_{P,V,W,\infty}$$

It follows from Lemma 3 that

$$\Pr(P \in \mathcal{S}_a | X_1 = V) - \Pr(P \in \mathcal{S}_a | X_1 = W) = \sum_{P \in \mathbf{End}(a): P(V) > 0} \Pr(P - V).$$

Below, we will define a set of size-n profiles (which we abbreviate as n-*profiles*), i.e. profiles with n votes $\mathcal{R}_n$, and prove the lower bound with respect to it. For a locally stable profile P^* (with constant γ in the statement of Definition 20), let $\vec{p}_0 = P^* - \vec{1} \cdot \frac{|P^*|}{m!}$. That is, $\vec{p}_0$ be obtained from P^* by subtracting a constant in each component, such that $\vec{p}_0 \cdot \vec{1} = 0$. For any n, we define $\mathcal{R}_n$ to be the set of n-profiles that are in the $\gamma\sqrt{n}$ neighborhood of $\frac{n}{m!} \cdot \vec{1} + \vec{p}_0 \cdot \sqrt{n}$ w.r.t. L_∞ norm for last $m! - 2$ dimensions. Formally:

$$\mathcal{R}_n = \left\{ P : P[V] = 0 \text{ and } \forall j \geq 3, \left| P[j] - \left(\frac{n}{m!} + \vec{p}_0[j] \cdot \sqrt{n} \right) \right| \leq \gamma\sqrt{n} \right\}$$

We next prove that the number of V votes in $\mathbf{Piv}(P)$ (defined in Sec. 3.4.2.1) and the number of W votes in $\mathbf{Piv}(P)$ are close—the difference is $O(\sqrt{n})$.

Claim 2. *For any $P \in \mathcal{R}_n$, we have $|\mathbf{Piv}(P)[V] - \mathbf{Piv}(P)[W]| = O(\sqrt{n})$.*

Proof. We use the hyperplanes $r = (H, g_H)$ notation for a GSR f^v. Let $Q^+ = \mathbf{Piv}(P)$ and $Q^- = \mathbf{Piv}(P) - V + W$. We note that $\mathbf{Piv}(P)$ is at the boundary of S, which means that $r(Q^+) \neq r(Q^-)$. Therefore, we have a GSR, the line segment between Q^+ and Q^- must contain the intersection of $\mathsf{T}_{P,V,W,\infty}$ and a hyperplane $\vec{h} \in H$. Therefore, it suffices to show that the difference in number of V votes and number of W votes at the intersection of $\mathsf{T}_{P,V,W,\infty}$ and any hyperplane $\vec{h}$ is $O(\sqrt{n})$.

We recall that by Lemma 6, all hyperplanes for r pass $\vec{1}$. For any $\vec{h} \in H$, we recall that we assumed that V and W corresponds to the first and second coordinate, respectively. Because $\vec{h} \cdot (P + l(V - W)) = 0$, we have $(h_2 - h_1)l = \vec{h} \cdot P = \vec{h} \cdot (P - \vec{1} \cdot \frac{n}{m!}) = O(\sqrt{n})$. This means that $|l| = |\mathbf{Piv}(P)[V] - \mathbf{Piv}(P)[W]| = O(\sqrt{n})$.

(End of Claim 2 proof)

□

Claim 3. *For any $P \in \mathcal{R}_n$, there is a V-W trail passing P.*

Proof. According to the canceling out property of r, we can construct profile $P' = P - \frac{n-|P^*|\sqrt{n}}{m!}$, which is equivalent to P. For any profile $P \in \mathcal{R}_n$, we have $\left|P[j] - \left(\frac{n}{m!} + \vec{p}_0[j] \cdot \sqrt{n}\right)\right| \leq \gamma\sqrt{n}$, which is equivalent with $|P'[j] - P^*[j] \cdot \sqrt{n}| \leq \gamma\sqrt{n}$ (recall $P[j]$ is the number of jth type of vote in profile P), which means $\frac{P'}{\sqrt{n}}$ is in the γ neighbourhood of profile P^* in terms of the 3-rd to $m!$-th dimensions. According to the (H, g_H) definition of GSR, we know $r(P^*) = r(P')$ and the claim follows by local stability of P^*.

(End of Claim 3 proof)

□

We will show that the probability of a subset of $\mathbf{End}(a)$—the pivotal profiles on trails starting at profiles in $\mathcal{R}_n$—is $\Theta(1/\sqrt{n})$ for the condition that π is uniform over $\mathcal{U}$. Let $\mathcal{R}_n^- \subseteq \mathbb{R}^{m!-2}$ and for any $\vec{p}_- \in \mathcal{R}_n^-$, we define $\mathbf{Piv}(\vec{p}_-) = \mathbf{Piv}(P)$, where $P \in \mathcal{R}_n$ and $P[3, \ldots, m!] = \vec{p}_-$.

$$\begin{aligned}
&\sum_{P \in \mathbf{End}(a)} \Pr(P - V) \geq \sum_{P \in \mathcal{R}_n} \Pr(\mathbf{Piv}(P) - V) \\
&= \sum_{\vec{p}_- \in \mathcal{R}_n^-, |P|=n-1} \Big(\Pr(P[3, ..., m!] = \vec{p}_-) \cdot \\
&\qquad \Pr(P[1] = \mathbf{Piv}(\vec{p}_-)[1] - 1, \Pr(P[2] = \mathbf{Piv}(\vec{p}_-)[2] | P[3, ..., m!] = \vec{p}_-) \Big) \\
&= \sum_{\vec{p}_- \in \mathcal{R}_n^-, |P|=n-1} A(\vec{p}_-) B(\vec{p}_-)
\end{aligned}$$

where $A(\vec{p}_-) = \Pr(P[3, \ldots, m!] = \vec{p}_-)$ and

$$B(\vec{p}_-) = \Pr(P[1] = \mathbf{Piv}(\vec{p}_-)[1] - 1, \Pr(P[2] = \mathbf{Piv}(\vec{p}_-)[2] | P[3, \ldots, m!] = \vec{p}_-)$$

It follows that $B(\vec{p}_-)$ is equivalent to probability of flipping a coin ($\frac{\pi[W]}{\pi[V]+\pi[W]}$ probability for head) for $\mathbf{Piv}(\vec{p}_-)[1] + \mathbf{Piv}(\vec{p}_-)[2] - 1$ times, with $\mathbf{Piv}(\vec{p}_-)[1] - 1$ heads and $\mathbf{Piv}(\vec{p}_-)[2]$ tails. The next lemma gives a lower bound to $\sum_{\vec{p}_- \in \mathcal{R}_n^-, |P|=n-1} A(\vec{p}_-) B(\vec{p}_-)$ when π is a uniform distribution.

Lemma 7. $\sum_{\vec{p}_- \in \mathcal{R}_n^-, |P|=n-1} A(\vec{p}_-) B(\vec{p}_-) = \Omega\left(\frac{1}{\sqrt{n}}\right)$ *if* π *is uniform over* $\mathcal{U}$.

Proof. We first bound the total number of V and W votes in $P \in \mathcal{R}_n$ in the next claim.

Claim 4. $\mathbf{Piv}(\vec{p}_-)[1] + \mathbf{Piv}(\vec{p}_-)[2] - 1 = \Theta(n)$ *for all* $\vec{p}_- \in \mathcal{R}_n^-$.

Proof.

$$\left|\mathbf{Piv}(\vec{p}_-)[1] + \mathbf{Piv}(\vec{p}_-)[2] - \frac{2n}{m!}\right| = \sum_{j=3}^{m!} \left|P[j] - \frac{n}{m!}\right| \leq \sum_{j=3}^{m!} \left(\gamma\sqrt{n} + |\vec{p_0}[j]|\sqrt{n}\right) \leq (\gamma + 1)m!\sqrt{n}$$

(End of Claim 4 proof)

□

According to Claim 2 & 4, we know that $B(\vec{p}_-)$ is equivalent to probability of flipping a fair coin for $\frac{2n}{m!} + c_1\sqrt{n}$ times and get $\frac{n}{m!} + c_2\sqrt{n}$, where c_1 and c_2 are bounded constants. In the next claim, we give a tight bound to $B(\vec{p}_-)$ for uniform distributed entries.

Claim 5. $B(\vec{p}_-) = \Theta\left(\sqrt{\frac{1}{n}}\right)$ *for any* $\vec{p}_- \in \mathcal{R}_n^-$

Proof. Letting $n' = \frac{2n}{m!} + c_1\sqrt{n}$, $c' = c_2 - \frac{c_1}{2}$ and assuming n' is a even number, for the lower bound, we have,

$$\begin{aligned} B(\vec{p}_-) &= \left(\frac{1}{2}\right)^{\frac{2n}{m!}+c_1\sqrt{n}} \binom{\frac{2n}{m!} + c_1\sqrt{n}}{\frac{n}{m!} + c_2\sqrt{n}} = \left(\frac{1}{2}\right)^{n'} \binom{n'}{n'/2 + c'\sqrt{n}} \\ &= \left(\frac{1}{2}\right)^{n'} \cdot \binom{n'}{n'/2} \cdot \frac{\frac{n'}{2} \times \cdots \times (\frac{n'}{2} - c'\sqrt{n'} + 1)}{(\frac{n'}{2} + c'\sqrt{n'} - 1) \times \cdots \times \frac{n'}{2}} \\ &> \frac{1}{2^{n'}}\binom{n'}{n'/2} \cdot \left(\frac{n'/2 - c'\sqrt{n'}}{n'/2}\right)^{c'\sqrt{n'}} \\ &= \Omega\left(\frac{1}{\sqrt{n}}\right) \quad \text{(applying Stirling's Formula)} \end{aligned} \tag{3.10}$$

Upper bound can be obtained using similar technique as lower bound.

(End of Claim 5 proof)

□

The next claim gives a lower bound on $\sum_{\vec{p}_- \in \mathcal{R}_n^-} A(\vec{p}_-)$. The proof uses the main technique of Lindeberg-Levy Central Limit Theorem [78].

Claim 6. $\sum_{\vec{p}_- \in \mathcal{R}_n^-} A(\vec{p}_-) = \Omega(1)$.

Proof of Claim 6. We first define a set of $m! - 2$ dimensions random variables that $Y_i = (Y_i[1], \cdots, Y_i[m! - 2])$, where $Y_i[j] = 1$ if ranking j happens to i-th row and $Y_i[j] = 0$ otherwise.

According to the definition of profile, we have $P[j+2] = \sum_{j=1}^{n} Y_i[j]$ and $\mathbb{E}(P[j]) = \frac{n}{m!}$ for uniform case. We further define a $m! - 2$ dimensional random vector $\vec{u}$ such that $\vec{u}[j] = \left(P[j+2] - \frac{n}{m!}\right)/\sqrt{n}$, which is the scaled average of $Y_1, \cdots, Y_n$. According to Lindeberg-Levy Central Limit Theorem [78], we know that the distribution of $\vec{u}$ converges in probability to multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where

$$\Sigma = \begin{bmatrix} \frac{m!-1}{(m!)^2} & -\frac{1}{(m!)^2} & \cdots & -\frac{1}{(m!)^2} \\ -\frac{1}{(m!)^2} & \frac{m!-1}{(m!)^2} & \cdots & -\frac{1}{(m!)^2} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{(m!)^2} & -\frac{1}{(m!)^2} & \cdots & \frac{m!-1}{(m!)^2} \end{bmatrix}.$$

Since each diagonal element in Σ is strictly larger than the sum of the absolute value of all other elements in the same row, we know that Σ is non-singular according to Levy-Desplanques Theorem [86]. According to Varah *et al.* [123], we obtain a bound on Σ^{-1}'s L_∞ norm as,

$$||\Sigma^{-1}||_\infty \leq \frac{1}{\min_i \left(|\Sigma_{ii}| - \sum_{j \neq i} |\Sigma_{ij}|\right)} \leq \frac{(m!)^2}{2}.$$

For any $m! - 2$ dimensional random vector $\vec{u}$ constructed from a profile P using the procedure that $\vec{u}[j] = \left(P[j+2] - \frac{n}{m!}\right)/\sqrt{n}$, we have,

$$P \in \mathcal{R}_n^- \quad \text{if and only if} \quad \vec{u} \in \mathbb{U} = \left\{\vec{u} : |\vec{u}[j] - \vec{p}_0[j]| \leq \gamma, \forall j \in [m! - 2]\right\}.$$

Thus, for all $\vec{u} \in \mathbb{U}$ we know about its Probability Density Function (PDF) that,

$$\begin{aligned}
\mathbf{PDF}(\vec{u}) &= \frac{1}{\sqrt{(2\pi)^{m!-2}|\Sigma|}} \exp\left(-\frac{1}{2}\vec{u}^T\Sigma^{-1}\vec{u}\right) \\
&= \frac{1}{\sqrt{(2\pi)^{m!-2}|\Sigma|}} \exp\left(-\frac{1}{2}|\vec{u}^T\Sigma^{-1}\vec{u}|\right) \\
&\geq \frac{1}{\sqrt{(2\pi)^{m!-2}|\Sigma|}} \exp\left(-\frac{1}{2}||\vec{u}^T\Sigma^{-1}||_\infty \cdot ||\vec{u}||_1\right) \text{ (Holder's Inequality)} \\
&\geq \frac{1}{\sqrt{(2\pi)^{m!-2}|\Sigma|}} \exp\left(-\frac{1}{2}||\vec{u}^T||_\infty \cdot ||\Sigma^{-1}||_\infty \cdot ||\vec{u}||_1\right) \\
&\geq \frac{1}{\sqrt{(2\pi)^{m!-2}|\Sigma|}} \left[\exp\left(\frac{(m!)^2}{4}\right)\right]^{-||\vec{u}||_\infty^2} \\
&= \Omega\left(1\right).
\end{aligned}$$

Thus, letting Vol$(\cdot)$ be the volume function,

$$\sum_{\vec{p}_- \in \mathcal{R}_n^-} A(\vec{p}_-) \geq \text{Vol}(\mathbb{U}) \cdot \min_{\vec{u} \in \mathbb{U}} \mathbf{PDF}(\vec{u}) \geq \gamma^{m!-2} \cdot \Omega\left(1\right) = \Omega(1).$$

(End of Claim 6 proof)

□

Lemma 7 follows be combining Claim 6 and Claim 5.

(End of Lemma 7 proof)

□

Let $P_1 = V$ mean to set one vote in profile P to V. Recalling Lemma 3, for the case that π is uniform over all rankings, we have,

$$\begin{aligned}\delta &= \max_{x,x',\mathcal{S}} \Pr(f^v(P) \in \mathcal{S}|P_1 = x) - \Pr(f^v(P) \in \mathcal{S}|P_1 = x') \\ &\leq \Pr(f^v(P) \in \mathcal{S}_a|P_1 = W) - \Pr(f^v(P) \in \mathcal{S}_a|P_1 = V) \\ &= \sum_{P \in \mathbf{End}(a)} \Pr(P - V) = \Omega\left(\frac{1}{\sqrt{n}}\right).\end{aligned}$$

Then, we derive an upper bound of δ using the similar technique to proving the lower bound ($\vec{\pi}$ can be non-uniform for this bound). We first give a definition of $\mathcal{R}'_n$, a subset of n-profile space (i.e. the space of profiles with n votes), where event $P \in \mathcal{R}'_n$ will be proved to happen with high probability. Formally,

$$\mathcal{R}'_n = \left\{P : P[V] = 0 \text{ and } \forall j \geq 3, |P[j] - (n \cdot \vec{\pi}[j])| \leq n^{3/4}\right\}.$$

Then, we recall Lemma 3, for the case that π such that $\min_i \pi[i] > 0$, we have,

$$\begin{aligned}\delta &= \max_{V,W,\mathcal{S}} \Pr(P \in \mathcal{S}|P_1 = V) - \Pr(P \in \mathcal{S}|P_1 = W) \\ &\leq \max_{V,W} \sum_{i=1}^{m} \Pr(P \in \mathcal{S}_i|P_1 = V) - \Pr(P \in \mathcal{S}_i|P_1 = W) = \sum_{i=1}^{m} \sum_{P \in \mathbf{End}(x_i)} \Pr(P - V).\end{aligned}$$

where $\mathcal{S}_i = \{P : r(P) = x_i\} = \bigcup_{P \in \mathbf{End}(x_i)} \mathsf{T}_{P,V,W,\infty}$.
The next claim gives an upper bound to $\sum_{\vec{p}_- \notin \mathcal{R}_n^-} A(\vec{p}_-)$.

Claim 7. $\sum_{\vec{p}_- \notin \mathcal{R}'^-_n} A(\vec{p}_-) = O\left(\frac{1}{\sqrt{n}}\right)$.

Proof. Let $Y_j^{(i)}$ = "the i-th agent gives vote of type j". One can see that $P[j] = \sum_{i=1}^{n} Y_j^{(i)}$, $\mathbb{E}(P[j]) = n\vec{\pi}[j]$ and $Var(P[j]) = n\vec{\pi}[j](1 - \vec{\pi}[j])$. Thus,

$$\begin{aligned}&\sum_{\vec{p}_- \notin \mathcal{R}_n^-} A(\vec{p}_-) = \Pr\left[\bigcup_{j=3}^{m!} \left\{\ |P[j] - n \cdot \vec{\pi}[j]| \leq n^{3/4}\right\}\right] \\ \leq &\sum_{j=3}^{m!} \Pr\left[\left\{\ \left|P[j] - \mathbb{E}(P[j])\right| \leq n^{3/4}\right\}\right] \\ \leq &\sum_{j=3}^{m!} \frac{n\vec{\pi}[j](1 - \vec{\pi}[j])}{n^{3/2}} \quad \text{(by Chebyshev's Inequality)} \\ = &O\left(\frac{1}{\sqrt{n}}\right)\end{aligned}$$

(End of Claim 7 proof)

□

Then, all we need is an upper bound on $B(\vec{p}_-)$, and we first prove that the length of $V - W$ sequence is $\Theta(n)$ for all $P \in \mathcal{R}'_n$.

Claim 8. $\mathbf{Piv}(\vec{p}_-)[1] + \mathbf{Piv}(\vec{p}_-)[2] - 1 = \Theta(n)$ *for all* $P \in \mathcal{R}'_n$.

Proof.

$$|\mathbf{Piv}(\vec{p}_-)[1] + \mathbf{Piv}(\vec{p}_-)[2] - n(\vec{\pi}[W] + \vec{\pi}[V])| = \sum_{j=3}^{m!} |P[j] - n \cdot \vec{\pi}[j]| \leq \sum_{j=3}^{m!} n^{3/4} \leq m! \cdot n^{3/4}$$

(End of Claim 8 proof)

□

The next claim gives am upper bound on the number of pivotal profiles sharing one End.

Claim 9. *For any profile* P *in* $\mathcal{R}'_n$*, there are at most* $|H|$ *pivotal profiles following* $V - W$ *direction.*

Proof. We know from the (H, g_H) definition of GSR that r's output only changes while passing at least one hyperplane. Considering a trail T_{P_0} enter at $(P_0[1] + P_0[2], 0, P_0[3], \cdots, P_0[m!])$ and exit at $(0, P_0[1] + P_0[2], P_0[3], \cdots, P_0[m!])$ (P_0 is an arbitrary n-profile). Thus, there are at most $|H|$ pivotal profiles sharing the same end point because T_{P_0} passes hyperplanes at most $|H|$ times.

(End of Claim 9 proof)

□

Using the partition of $\mathcal{R}'_n$ and arbitrarily selected candidate a, we have,

$$\begin{aligned}
\sum_{P \in \mathbf{End}(x_i)} \Pr(P - V) \leq & |H| \left(\sum_{P \in \mathcal{R}'_n} \Pr(\mathbf{Piv}(P) - V) + \sum_{P \in \mathbf{End}(x_i) \setminus \mathcal{R}'_n} \Pr(\mathbf{Piv}(P) - V) \right) \\
\leq & |H| \left(\sum_{\vec{p}_- \in \mathcal{R}'^{-}_n, |P| = n-1} A(\vec{p}_-) B(\vec{p}_-) + \sum_{\vec{p}_- \notin \mathcal{R}'^{-}_n, |P| = n-1} A(\vec{p}_-) B(\vec{p}_-) \right) \\
\leq & |H| \left(\max_{\vec{p}_- \in \mathcal{R}'^{-}_n} B(\vec{p}_-) \cdot \sum_{\vec{p}_- \in \mathcal{R}'^{-}_n} A(\vec{p}_-) + \max_{\vec{p}_- \notin \mathcal{R}'^{-}_n} B(\vec{p}_-) \cdot \sum_{\vec{p}_- \notin \mathcal{R}'^{-}_n} A(\vec{p}_-) \right) \\
= & O\left(\frac{1}{\sqrt{n}}\right) \cdot O(1) + O(1) \cdot O\left(\frac{1}{\sqrt{n}}\right) \quad \text{(By applying the above Claims)} \\
= & O\left(\frac{1}{\sqrt{n}}\right)
\end{aligned}$$

Then, using the same technique of Claim 5, we know that,

$$B(\vec{p}_-) = \Theta\left(\sqrt{\frac{1}{n}}\right) \quad \text{for all} \quad p_- \in \mathcal{R}_n'^-$$

Thus, combining all results above, we have,

$$\delta \leq \sum_{i=1}^{m} \sum_{P \in \mathbf{End}(x_i)} \Pr(P - V) = \sum_{i=1}^{m} \sum_{P \in \mathbf{End}(x_i)} \Pr(P - V) = O\left(\frac{1}{\sqrt{n}}\right)$$

Next, we will give a exponential (tighter) upper bound on δ when $\vec{\pi}$ does not belong to any hyperplanes. We first give a generalized definition of a pivotal profile.

Definition 24 (Generalized pivotal profile)**.** A profile P is a *(generalized) pivotal profile* if there exist pair of votes V and W such that $r(P) \neq r(P - V + W)$.

Then, we define a distance function $\mathbf{Dist}^*(P, h)$ to be a generalized distance between profile P and hyperplane h. We define

$$\mathbf{Dist}^*(P, \vec{h}) = \inf_{P' \in \tilde{h}} ||P - P'||_2,$$

where $\tilde{h} = \{P \in h : \exists \text{ unit vector } \vec{e} \text{ s.t. } r(P' - \vec{e}) \neq r(P' + \vec{e})\}$. In the next lemma we will show generalized pivotal profiles only lays close to hyperplanes. We also also define the distance function $\mathbf{Dist}(\cdot, \cdot)$ below:

1. For hyperplane h and a point (n-profile) P, $\mathbf{Dist}(h, P) = \frac{\vec{h} \cdot P}{||\vec{h}||_2}$, which is the Euclidean distance between P and hyperplane $\vec{h} \cdot \vec{p} = 0$.
2. For 2 points (n-profiles) P_1 and P_2, $\mathbf{Dist}(h, P)$ returns the Euclidean distance between P_1 and P_2.

Claim 10. *For any GSR represented by $r = (H, g_H)$ and one of its generalized pivotal profile P, there must exist one hyperplane $\vec{h} \in H$ such that $\mathbf{Dist}(h, P) \leq \sqrt{2}$.*

Proof. Recalling the definition of generalized pivotal profiles, we know the GSR winner will change at the 1 neighborhood of P. Thus, there must exist a hyperplane $\vec{h} \in H$ and pair of votes V, W such that $\mathbf{Sign}\left[\vec{h} \cdot P\right] \neq \mathbf{Sign}\left[\vec{h} \cdot (P + V - W)\right]$ and $\mathbf{Dist}(h,\ P) \leq \mathbf{Dist}(P, P + V - W) = \sqrt{2}$.

(End of Claim 10 proof)

□

Lemma 8. *Let D be the distribution on profiles (databases of votes), where each entry is iid according to distribution $\vec{\pi}$ over linear orders on m candidates. Then GSR $r(H, h_H)$ is $(0, \delta, \Delta = \{(D, \emptyset)\})$-DDP when only the winner is announced, where*

$$\delta = O\left[\exp\left(-\frac{\left[\min_{h\in H} \mathbf{Dist}^*(\vec{\pi}, h)\right]^2}{3(m!)^2\left(\max_{i\in[m!]}\vec{\pi}[i]\right)}\cdot n\right)\right] = O\left[e^{-\Omega(n)}\right].$$

Proof. We first define the set of all generalized pivotal profiles $\mathbb{P}_{\mathsf{Piv}}$. For any $P \in \mathbb{P}_{\mathsf{Piv}}$, we know that there exist hyperplane $h \in H$ such that $\mathbf{Dist}^*(h,\, P) \leq \sqrt{2}$. According to triangular inequality, we have $\mathbf{Dist}^*(n\vec{\pi},\, P) \geq \mathbf{Dist}^*(n\vec{\pi},\, h) - \mathbf{Dist}(h,\, P) \geq n\mathbf{Dist}^*(\vec{\pi},\, h) - \sqrt{2}$. The second $\geq$ sign comes from the fact that all hyperplanes passes $\vec{0}$. Thus, there must exist one dimension j that $|P[j] - n\vec{\pi}[j]| \geq \frac{n\mathbf{Dist}^*(\vec{\pi},h)-\sqrt{2}}{m!}$. Then, we bound δ as,

$$\begin{aligned}
\delta =& \max_{V,W,\mathcal{S}} \left[\Pr(P \in \mathcal{S}_i | X_1 = V) - \Pr(P \in \mathcal{S}_i | X_1 = W)\right] \\
\leq& \sum_{P\in\mathbb{P}_{\mathsf{Piv}}} \left[\max_{V} \Pr(P \in \mathbb{P}_{\mathsf{Piv}} | X_1 = V)\right] \\
\leq& \max_{V,h,j} \Pr\left(|P[j] - n\vec{\pi}[j]| \geq \frac{n\mathbf{Dist}^*(\vec{\pi},\, h) - \sqrt{2}}{m!} \middle| X_1 = V\right) \\
\leq& \max_{h,j} \Pr\left(|P[j] - n\vec{\pi}[j]| \geq \frac{n\mathbf{Dist}^*(\vec{\pi},\, h) - \sqrt{2}}{m!} - 1\right) \\
=& O\left[\exp\left(-\frac{\left[\min_{h\in H} \mathbf{Dist}^*(\vec{\pi}, h)\right]^2}{3(m!)^2\left(\max_{i\in[m!]}\vec{\pi}[i]\right)}\cdot n\right)\right] \text{ by applying Chernoff bound.}
\end{aligned}$$

(End of Lemma 8 proof)

□

Theorem 4 follows by combining all three bounds derived above.

(End of Theorem 4 proof)

□

3.4.3 Exact privacy: GSR examples

We first use a simple example of the plurality rule (i.e. 0.5-biased majority rule) to show the results in Theorem 4 matches the 2-candidate results in our previous section.

Example 4 (Example of Definition 7 and Theorem 4)**.** Let $\mathcal{U} = \mathcal{C} = \{c_1, c_2\}$, $V = [c_1 \succ c_2]$, and $W = [c_2 \succ c_1]$. For the *α-biased majority rule* with $\alpha = 0.5$ (i.e. plurality rule), we have $f^{gsr}(V) = (1, 0)$ and $f^{gsr}(W) = (0, 1)$. Then, the winner is chosen according to

g^{gsr} corresponding to the largest component in $f^{gsr}(P)$. Recalling our definition of unstable distribution, we know $(\frac{1}{2}, \frac{1}{2})$ is the only unstable distribution for 2-candidate majority rule. This is the intuitive reason behind $\delta = \Theta(\sqrt{1/n})$ when $\pi = (\frac{1}{2}, \frac{1}{2})$ for both Theorem 4 and Theorem 2 (when $\alpha = 0.5$). For any other $\pi \neq (\frac{1}{2}, \frac{1}{2})$, these two theorems result in $\delta = \exp[-\Omega(n)]$. We note that while Theorem 4 covers more voting rules, Theorem 2 is a more fine-grained result for two candidates.

Corollary 2. *Plurality, veto, k-approval, Borda, Maximin, Copeland, Bucklin, Ranked Pairs, Schulze (see e.g. [127]) are $(0, \Theta\left(1/\sqrt{n}\right), \Delta)$-eDDP when Δ contains the uniform distribution.*

Proof. As shown in Definition 20, *canceling-out* and *monotonicity* are very natural properties of most voting rules. These two properties can be easily checked according to the definitions of voting rules discussed in Corollary 2. In the next proposition, we prove a more generalized version of Corollary 2 for *local stability*, which indicate a large subset of the voting rules can satisfies all properties required by Theorem 4.

Proposition 1. *All positional scoring rules and all Condorcet consistent and monotonic rules satisfy the property of* local stability*.*

Proof. Let s_i to denote the score of the i-th candidate, i.e.the ith component of $f^{gsr}(P)$ for a profile P. Suppose $s_1 = \cdots = s_l > s_{l+1}$. We let $V = [a \succ c_1 \succ c_{l-1} \succ b \succ$ others$]$ and $W = [c_1 \succ c_{l-1} \succ b \succ a \succ$ others$]$. Let M be the permutation $c_1 \to c_2 \to \ldots c_{m-2} \to c_1$. Let $V_1 = [a \succ b \succ$ others$]$ and $V_2 = [b \succ a \succ$ others$]$. Let $P' = \bigcup_{i=1}^{m-2} M^i(V_1) \cup M^i(V_2)$. Let $P^* = 2P' \cup \{V, W\}$. It follows that a and b are the only two candidates tied in the first place in P^*. Therefore, there exists ϵ to satisfy the condition in local stability.

The same profile can be used to prove the local stability of all Condorcet consistent and monotonic rules. □

Then, Corollary 2 follows by combining the results for all three properties.

□

Another commonly-used GSR called STV does not satisfy monotonicity, which means that Theorem 4 does not apply. However, empirical results (Section 3.5) suggest that STV is likely also $(0, \Theta\left(1/\sqrt{n}\right), \Delta)$-eDDP for Δ containing the uniform distribution.

3.5 Concrete estimation of privacy parameters

We present an example of computing concrete estimates of $(0, \delta, \Delta)$-exact DDP values for several GSRs. For this example, we let $\Delta = \{\pi\}$ such that $\pi \in \Pi(\{x_1, x_2, x_3\})$ and $\pi(x_i) = \pi(x_j) = 1/3$ (i.e., votes are i.i.d. and uniform).

We generated these concrete estimates via exhaustive search over possible profiles for 3 candidates and $n \leq 50$ votes, and computing the δ values exactly for each n. Since we know that $\delta = \Theta(1/\sqrt{n})$, we fit these values to $\delta(n) = \frac{1}{\sqrt{an+b}}$ via linear regression. We rank voting rules from most to least private. The larger the a, the smaller the δ value and thus more private:

Rule	Winner	Mean Square Error ($n \in [50]$)
Borda	$\delta(n) = \frac{1}{\sqrt{1.347n + 0.5263}}$	0.0566844201243
STV	$\delta(n) = \frac{1}{\sqrt{1.495n + 0.02669}}$	0.0542992943035
Maximin	$\delta(n) = \frac{1}{\sqrt{1.553n + 4.433}}$	0.0377631805983
Plurality	$\delta(n) = \frac{1}{\sqrt{1.717n - 0.09225}}$	0.0477175838906
2-approval	$\delta(n) = \frac{1}{\sqrt{1.786n + 0.3536}}$	0.0454223047191

Table 3.2: δ values in $(0, \delta, \Delta)$-eDDP for some commonly-used voting rules under the i.i.d. uniform distribution. $m = 3$ and $n = 10$ to 50.

2-approval $\triangleright$ Plurality $\triangleright$ Maximin $\triangleright$ STV $\triangleright$ Borda

We show in Table 3.2 the fitted δ curves with the mean square error in the fit.

Figure 3.3 shows the comparison between Plurality, Borda, and STV voting rules w.r.t. their δ values in $(0, \delta, \Delta)$-eDDP, when fitted to $\delta(n) = \frac{1}{\sqrt{an+b}}$.

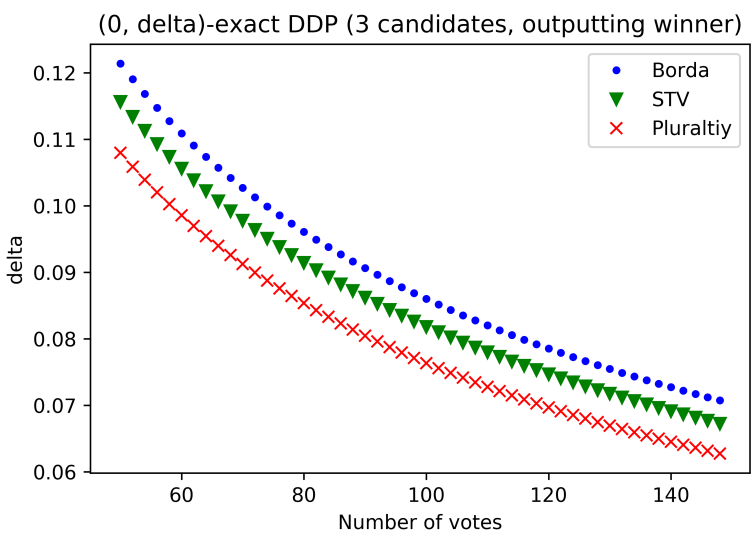

Figure 3.3: The δ values in $(0, \delta, \Delta)$-eDDP for Borda, STV, and plurality in our concrete estimates.

3.6 Chapter summary and future work

We address the limitation of DP in deterministic voting rules by introducing and characterizing (exact) DDP (eDDP) for voting rules, leading to an encouraging message about the good privacy of commonly-used voting rules and a framework to compare them with respect to eDDP. There are many directions for future work. An immediate open question for theoretical study is to extend our studies to general (ϵ, δ), and non-i.i.d. distributions, as well as to other high-stakes social choice procedures such as matching and resource allocation. On the practical side, it could be informative to study the eDDP of other data that is often published during an election, such as demographic information, and interpret their consequences.

Chapter 4

Cryptographic Implementation of Collusion-Free and Preserving Games

4.1 Introduction

Subliminal communication channels in protocols allow parties to embed extra information into protocol messages, often without being detected. The existence of subliminal channels is problematic in several applications of secure computation. In large-scale distributed systems, for instance, subliminal channels could allow two parties to coordinate their actions (i.e., collude) even if they may not have been aware of each other in advance. Such collusions have severe consequences in game theoretic applications, where stability, e.g., Nash equilibrium, is defined in terms of isolated strategies. An example is the prototypical application of distributed cryptography, namely, playing poker in a distributed manner [75]. An MPC protocol which allows collusions changes the rule of the game —think of playing poker against colluding opponents.

In the quest to combine game theory and cryptography, a number of works [8, 11, 41, 99, 100] put forth new security notions, and in particular the notion of *collusion-freeness (CF)*. Analogous to simulation-based security, where a protocol is "secure" if the view of the adversary (controlling corrupt parties) can be emulated by a simulator, a protocol is *collusion-free*, if the view of individual corrupt parties can be emulated by individual non-colluding simulators. However, collusion-freeness is impossible when parties are connected by pairwise communication channels—this is straightforward to see since such channels can directly be used for coordination.

The impossibility of collusion-freeness under pairwise channels led to the proposal of alternative models that enable this desirable property. The two most typical models are (1) assuming players are physically collocated, have access to a semi-trusted "ballot box," and can communicate publicly via (physical) envelopes [99, 100], or (2) assuming that parties are *only* connected to a semi-trusted party (via standard communication channels) called the *mediator* [8, 11] in a star network topology. In each round, the mediator is trusted to correctly perform a two-party computation with each party individually. Roughly, the goal of this computation is to remove any

Most of the results in this chapter have been published in [47].

embedded subliminal communication in the protocol messages.

Unfortunately, collusion-freeness, as a standalone definition, is not enough to limit collusion when parties can be engaged in multiple protocols. As argued by Alwen et al. [9], unlike what one might expect, a CF protocol (secure according to the definition of [8, 11]) does not necessarily preserve its "collusion-freeness" when composed with other protocols. Alwen et al. gives the following simple counter-example. Suppose a CF protocol is augmented with the following: it allows two parties A and B to collude only if party B can provide the correct (randomly-generated at run-time) κ-bit password, but the password is given only to party A. The protocol remains CF, since B can only guess the password with negligible probability. However, if through executing another protocol, party A can communicate κ-bits to B, then A can send the password and collusion-freeness is lost. This limitation motivated [9] to introduce the notion of *collusion-preserving* computation (CP). Intuitively, this notion can be seen as a universally composable (UC) [36] extension of CF, designed to explicitly address the above issue. (An alternative model capturing collusion-preserving universally composable computation via local adversaries was concurrently and independently proposed by Canetti and Vald [41].) As a concrete example, suppose a protocol Π is CP, and suppose while executing Π, a corrupt party p_i uses an external communication channel to send another party p_j a message m (e.g. a secret value only p_i knows). CP ensures that knowing m does not allow p_j to gain *even more* information in Π, such as p_i's input or output, other than information already implied by m. Importantly, CP supports composition and it is modeled with respect to arbitrary communication resources.

Continuing the line of works in the mediated model, [9] constructed a collusion-preserving (CP) protocol, which achieves the following fallback guarantee: Assuming a global augmented common reference string (ACRS) functionality [39] [1] , when parties are arranged in a star topology with the mediator in the center, there exists a protocol which (1) CP emulates any given functionality if the mediator is honest, and (2) remains GUC secure [39]—i.e., secure with abort according to the monolithic-adversary definition—even if the mediator gets corrupted (though CP/CF is lost).

On the downside, the CP protocol of [9] in the mediated model presents several limitations. First, it is straightforward to prove that desirable properties such as fairness and identifiable abort are impossible in the mediated model when the mediator might get corrupted (Sec. 4.5.1). This can lead to undesirable situations—such as a poker tournament where a player can always abort the game when he realizes he is losing, without being identified as a cheater. Second, the protocol (compiler) from [9] takes explicit steps to ensure that an abort does not allow parties to correlate their strategies, by making sure that an abort is observable only in a final round that is deterministically fixed at the beginning of the protocol. However, this means that their solution cannot be used to compute reactive functionalities [2] Lastly, the protocol is not round-efficient. Having a deterministic upper bound on the number of rounds means that, if the

[1] In strategic games, players may achieve additional (correlated) equilibria by observing the same public signal, such as the ACRS in [9] and protocol messages in our construction. However, CP ensures that no meaningful information (about the input or output) can be communicated using the additional correlation.

[2] In [100] it is observed that given a CF protocol for non-reactive games, avoiding collusion for reactive games is trivial if they have *short* descriptions. That is, if players commit their entire strategies at the beginning, then any collusion during the game does not affect the outcome. While situation is more complicated if malicious parties aborts and stops the game, we solve this issue by disincentivizing penalization, e.g. Section 4.4.

goal is to unanimously decide on whether or not an abort occurred even in the fallback setting where the mediator is compromised, the protocol always needs a linear number of rounds. This is true because a generic compiler for such a functionality would imply deterministic broadcast which needs linearly many rounds [55]. In fact, in the [9] compiler, each round is emulated by a round-robin sequential interaction of each party with the mediator, yielding an additional linear multiplicative blowup in the round complexity.

In this chapter, we will circumvent several of the limitations of [9] and extend its applicability towards a framework for collusion-preserving protocols over public networks. To our knowledge, this is the first work proposing a solution that breaks the deadlock of collusion-free/preserving computation, which was believed to only apply to the mediated model or require physical presence of parties in the same room.

Concretely, our solution replaces the mediator by the strictly weaker (as we argue later) assumption of an authenticated broadcast channel and honestly generated hardware tokens. As we will show (Sec. 4.1.3): (1) Capturing such hardware tokens in a collusion-preserving framework is non-trivial (2) Our protocols achieve CP when the adversary does not abort (and in fact CP is impossible with the broadcast channel when the adversary can abort). To disincentivize aborts, we show how to leverage the publicly identifiable abort (Section 4.2.1) property of our protocol to concretely penalize (e.g., via the blockchain) aborting parties. The fact that our protocol is CP means that any external communication introduced by the blockchain will not lead to even more correlation in our protocol. To capture incentive-driven attackers in a composable manner, we combine and extend the CP framework with the *rational protocol design* (RPD) methodology [65], which we introduced in the preliminaries Ch. 2, Sec. 2.6.2. We believe that both our treatment of hardware tokens in CP, and our incentives model which we term RPD-CP, can be of independent interest.

As fallback in case the hardware tokens may be compromised, our proposed protocol protects the inputs of honest parties and ensures identifiable (unanimous) [3] abort while guaranteeing termination (cf. [91]). This is the analogue—in the token-hybrid setting with a broadcast channel—of the fallback property of [8,9,11] which preserves privacy against a corrupted mediator at the center of a a star-network. In fact, our fallback is stronger than what the mediated-model permits [9]; indeed, the star-network topology makes it impossible to obtain identifiable (unanimous) abort against a corrupted mediator. The reason is that since the mediator controls the communication, it is impossible to correctly detect if a malicious party did not sent a message, or if it is the malicious mediator is pretending that an honest party has stopped replying (Sec. 4.5.1).

Chapter organization. Sec. 4.1.1: Overview of our contributions. Sec. 4.1.2: Related literature. Sec. 4.1.3: Overview of our constructions and main techniques. Sec. 4.2: Protocol Π^{HT} for CP functionalities assuming tokens, broadcast and no abort. Sec. 4.2.3: Protocol $\Pi^{\text{HT-FBS}}$ with fallback GUC security against compromised tokens. Sec. 4.3: New framework RPD-CP for defining CPAP—security of CP protocols against rational attackers. Sec. 4.3.3: Proofs of

[3] Recall from preliminaries Sec. 2.4.1.3 that security with *unanimous abort* guarantees that either all or none of the honest parties receive the output while *identifiable abort* ensures any party causing an abort can be identified (and excluded from future executions).

CPAP security for our protocols Π^{HT} and $\Pi^{\text{HT-FBS}}$. Sec. 4.4: Penalization scheme to make the utilities defined in Sec. 4.3 concrete.

4.1.1 Overview of our contributions

We now present our contributions in more detail. The goals of this chapter are to construct CP protocols that (1) replace the mediator-centered star-topology network by weaker resources which are closer to modern communication networks, (2) achieve stronger fallback security properties and (3) ensure CP computation of even reactive functionalities by proving that incentive-driven attackers will not abort, in a security model that incurs the negative cost of abort to the adversary. These goals bring the theory of CP closer to capturing real world applications such as a decentralized-dealer poker game. For reference, a comparison of our results to [100] and [8, 9] can be found in Table 4.1. In more detail, our protocol emulates CP functionalities, including those with private actions, when no abort occurs. We disincentivize aborts via a concrete penalization scheme and in addition, we achieve fallback security maintaining identifiable abort (and unanimous abort). That is, even when hardware tokens are compromised and the parties are allowed to abort we still retain standard GUC security. The abort column in the table specifies the number of bits of subliminal communication possible when an abort happens.

As a first step towards our goals, we show how to construct a collusion-preserving protocol Π^{HT} allowing n parties to CP emulate any given what we call *CP-well-formed functionality* which, when everyone gets corrupted, give up on collusion-preservation.

Definition 25 (CP-well-formed functionality)**.** We say that a functionality $\mathcal{F}$ is a *CP-well-formed functionality* if, when all parties are corrupt $\mathcal{F}$ has the following behavior on its adversaries' interfaces:

1. Whenever a message m is received on the ith adversarial interface, $\mathcal{F}$ outputs (i, m) to the first adversarial interface.

2. Whenever a message of the form $(i, \texttt{msg})$ is received on the first adversarial interface, $\mathcal{F}$ outputs the message m to the ith adversarial interface.

Our protocol uses as resources (1) honestly generated stateful trusted hardware tokens (HTs) and (2) an authenticated broadcast channel available during protocol execution[4]. Our protocol, which can CP-emulate any functionality as long as no abort occurs, improves upon the CF-protocol of [100], which is also based on broadcast but is not CP and does not emulate games with private actions (i.e. actions that are not publicly observable). We note that our protocol, analogously to that of [100], remains (G)UC secure with identifiable abort (though not collusion-free) in case of abort. We remark that Π^{HT} only requires two (broadcast) communication rounds unlike that of [100] or [9]. Furthermore, unlike the mediator from [9], tokens do not need to know in advance what computation they will be used for, and only need to be initialized with correlated randomness independent of the protocol, discussed in the overview of our techniques.

[4]The broadcast channel we employ guarantees that messages are always delivered to the parties [64, 85].

The protocol Π^{HT} only offers security guarantees when the tokens are uncompromised. This is arguably a strong assumption since in reality the adversary may attempt to break the security of the tokens. For this reason we present a protocol compiler $\Pi^{\text{HT-FBS}}$ which on input a (standard (G)UC secure) protocol with *unanimous* or *identifiable abort*, outputs a protocol with the same CP guarantees as Π^{HT} and, additionally, preserves (G)UC security properties (i.e., security with *unanimous* or *identifiable abort*, respectively) of the compiled protocol, even when hardware tokens are compromised. Specifically, even if the memory/code of corrupt parties' tokens can be read/reprogrammed and the secret keys of honest parties' tokens are leaked, the protocol is still (G)UC secure. This improves upon the fallback security of [9] where, due to model idiosyncrasies, these properties are impossible to achieve when the mediator is corrupted, even with honest majority (also in this case the reason is that the mediator has full control of the network, see Sec. 4.5.1). We note that $\Pi^{\text{HT-FBS}}$, even with the extra fallback guarantee, has round complexity that is still lower than the mediated-model solution of [9].

We believe that the above corruption model for the fallback solution is quite realistic, and captures most of the attacks that have been performed on hardware tokens. For this reason, our main theorems are proven in this corruption model. However, for sake of completeness, we will follow with a sketch of how to construct a protocol that preserves (G)UC security even in the case where also the tokens of the honest parties are fully compromised (i.e., the memory/code of any token can be read/reprogrammed). However, this protocol will require a higher communication complexity, and we lose the property of identifiable abort.

As an additional contribution, we combine the Rational Protocol Design (RPD) [15,65] framework (see Preliminaries Section 2.6.2)with CP and define a new model termed *RPD-CP*. Within this model we define a CP-security notion *collusion-preserving attack-payoff* (in short, *CPAP*), which intuitively corresponds to security against any combination of incentive-driven local adversaries. We identify a natural class of utilities under which non-aborting strategies are strictly dominant in Π^{HT} and $\Pi^{\text{HT-FBS}}$. That is, these protocols are collusion-preserving according to CPAP against adversaries bounded by this utility. We believe that RPD-CP can be used to derive formal composable versions of security statements with fair-compensation [12, 24, 93–95] which we think is an interesting future direction. Finally we propose a concrete penalization mechanism, which may be implemented via e.g., the blockchain, that induces utilities in the above class. Combining this with the notion of CPAP, we can prove that any adversary who maximizes its revenue (or at least minimizes loss) will not abort, making our protocol CP secure against such adversaries. The ability of the CP security definition to make formal statements in the presence of a global blockchain demonstrates the power of the CP definition. Indeed, CP ensures that the protocol participants cannot abuse the external channel provided by the blockchain to increase the number of bits of collusion. That is, suppose one devises an equilibrium in an *ideal* poker game (with a trusted dealer) where the players can also access the blockchain. Then, by adapting the results from [9], we can prove that this equilibrium would be preserved when the poker dealer is replaced by our CP protocol.

	Channel, assumption	Id-abort (fallback)	U-abort (fallback)	Private actions	Abort
Lepinski et al., STOC 2005	Broadcast+physical presence of parties, physical envelopes	✓	✓	✗	poly(κ) bits
Alwen et al., Crypto 2009 **Alwen et al., Crypto 2012**	Star topology, honest mediator	✗	✗	✓	$\Omega(\log \kappa)$ bits for reactive functionalities
This work	Authenticated Broadcast, hardware tokens	✓	✓	✓	Disincentivized

Table 4.1: Comparison with existing approaches. Id-abort: identifiable abort, U-abort: unanimous abort.

4.1.2 Related works

4.1.2.1 Collusion-freeness and preservation

We extend the comparison of our results with existing results on collusion-freeness (CF) and preservation (CP). The work of Lepinski et al. [100] achieves collusion-freeness for parties that communicate with a broadcast channel, assuming access to a physical primitive called "envelopes". They motivate the use of envelopes by proving that with only (authenticated) broadcast channels, CF is impossible, even when the adversary does not cause the computation to abort. The idea is that corrupted parties can share the same random tape before the start of protocol execution. Since all messages are sent through broadcast, all corrupted parties will have the same view and thus emulate a monolithic adversary. In the setting of our results, parties also communicate via a broadcast channel, but we circumvent this impossibility since the random tape of a corrupted party is not decided by the party himself, but by a hardware token (whereas Lepinski et al. generate randomness through coin-tossing and hide the result in envelopes).

Using stateful hardware tokens, we can also circumvent another impossibility result of Lepinski et al., that CF protocols with private actions/inputs are not possible even with envelopes. The impossibility comes from the corrupted parties being able to see (different) protocol messages that different private actions generate, such as the result of encryption on different inputs. The corrupted party then may be able to choose his private action, such that the resulting protocol messages convey subliminal information about the private action itself (e.g., the last bits of the protocol message encodes the action). However, this can be circumvented by stateful hardware tokens. Informally, tokens can be programmed to prevent users from changing his input and ensure they cannot see messages generated by different inputs. Thus, our protocol remains CP even with private actions.

Lastly, as described in more detail in the introduction, in the mediated model, [8,11] achieve CF and [9] achieves CP. In [41] the authors consider the notion of UC security with local adversaries, which is more general than the notion of CP as it captures more complex corruption models. In particular, [41] captures the scenario where there are independent clusters of corrupted parties, whereas CP assumes that each corrupted party works in isolation (though their code is generated by a single malicious attacker).

4.1.2.2 Authenticated broadcast assumption

We compare authenticated broadcast with assumptions from previous works. Existing works on CF and CP either require parties to be present in person [100]—to pass around physical envelopes, or rely on restricted star-topology communication networks [8, 9]. These assumptions are used far less than the (authenticated) broadcast channel common in MPC literature, and have been criticized as overly restrictive and/or unrealistic.

In particular, the physical presence assumption of [100] makes the resulting protocols inapplicable to the standard cryptographic setting, where the protocol is played by interactive Turing machines (ITMs). Similarly, the mediated model [8, 9] requires both complete isolation of the parties and a special star-network topology where the mediator (at the center) is required to both preserve the privacy of the communication and generate (pseudo)randomness—these requirement are proven to be necessary for CP in the mediated model [9]. This is in contrast to the less demanding assumption of authenticated broadcast combined with honestly generated hardware tokens, which not only we believe is more natural—authenticated broadcast is a standard assumption in the cryptographic protocols literature and trusted hardware is a far less exotic assumption than it used to be, but is also formally weaker as discussed in Section 4.1.2.4.

As in previous works, if our communication resource (i.e. broadcast) is malicious (e.g., allows undetectable communication between corrupt parties) then our protocols are no longer CP. (In fact it is easy to verify that such undetectable communication would make CP infeasible.) Nonetheless, analogous to "fallback" security in the CP setting with a corrupted mediator [9], our protocol still preserves its standard (G)UC security guarantees under a malicious broadcast resource (see Section 4.2.3.2).

4.1.2.3 Playing games over the blockchain

The question of playing games such as poker over the Internet has recently attracted considerable attention, fueled by the new capabilities introduced by smart-contract-enabled (cryptocurrency) blockchains, such as Ethereum. In a nutshell, this technology makes it possible to ensure that parties cannot avoid paying their bid amount when they lose without the need of a trusted escrow—by having them commit their bids on the blockchain, in a smart contract that releases them to anyone that presents evidence of winning. Furthermore, the same technology enables a mechanism that punishes cheating—or early aborting—by making parties commit collateral that they can only claim if evidence is presented that they completed their protocol. This method of penalization has been used by, e.g., [12, 24, 93–95], which gave rise to a number of proposals for decentralized poker protocols [25, 96]. However, all these works use standard multi-party computation, thus even players who do not know each other can collude via protocol messages.

4.1.2.4 Stateful tamper-resilient hardware tokens

Previous works (e.g. [10,52,56,74,77,90,109,114]) have based (UC-)secure protocols on stateful, tamper-resilient hardware. In addition to theory, trusted hardware such as Intel's SGX ([19, 20, 63]) and Bitcoin Hardware Wallets, have also been deployed in real life. The protocol of Lepinski et al. [100] is based on physical envelopes, a kind of trusted hardware as well. They constructed

CF (though not CP) protocols for games with public actions, where parties pass around such envelopes.

We extend the application of hardware tokens by presenting the *first* (in our knowledge) collusion-preserving protocol based on stateful trusted hardware tokens. We emphasize that while improving upon the limitations and impossibilities of previous works, we do not remove the need for trust completely. However, instead of relying on a trusted mediator to achieve CP, we replace it with authenticated broadcast (discussed in Section 4.1.2.2) and trusted hardware tokens. Intuitively, these assumptions are weaker than the mediator, since an honest mediator in a star topology can act as a broadcast channel and as a set of hardware tokens. In fact in Sec. 4.5 we prove a formal separation of the assumption of a trusted mediator from the one used here (i.e., the combination of broadcast and honestly generated hardware tokens) demonstrating that our assumption is indeed strictly weaker than that of a trusted mediator; in particular we show that a trusted mediator allows for CP even in the presence of an aborting adversary, which is impossible in our setting.

With trusted tokens, we improve upon the solution of [100] by achieving composition, games with private actions, and only requiring parties to broadcast messages instead of physically exchanging tokens. We also circumvent the impossibilities of previous works in collusion-preservation, as discussed in the introduction. To improve the practicality of our solutions, in Section 4.2.3 we propose protocols that provide fallback security in case of compromised tokens (we discuss two levels of severity of compromise), as well as address practical issues in implementation (Section 4.2.2).

Below, we detail the properties of our token assumption. We follow the approach of [10] to model hardware tokens as ideal functionalities. Our tokens require the following properties:

- *Stateful*: The token has internal memory which may be read/updated.
- *Trusted and Tamper-resilient*: The token manufacturer is trusted, and no one except the token itself can read or write its contents. In Section 4.2.3 we also sketch solutions (with some trade-offs) which preserve GUC security given untrusted or non-tamper-resilient tokens.
- *Isolated* from its creator: Only the token owner can query and get the outputs generated by the token. In particular, we require that no other parties may communicate with the token.

The formalism of isolated, tamper-resilient and stateful hardware tokens was introduced by [90]. There, the creation process of a token is described by a "wrapper" functionality which allows parties to store and run (possibly several) ITMs representing the code and memory of tokens. The wrapper functionality models malicious token creators in the protocol. However, to achieve CP, we assume that all the parties hold honestly-generated tokens that share some common private information. Thus, following [10], we model each token as a ideal functionality without using the wrapper. Proving collusion-preservation based on hardware tokens presents several unique technical challenges. We detail these issues and their solutions in Section 4.1.3.

4.1.3 Overview of our techniques

Let $\mathcal{P}$ be a set of parties who wish to compute a function f in a collusion-preserving way. We assume that each party $p_i \in \mathcal{P}$ has access to a hardware token (HT) $\mathtt{HT}_i$. All HT's contain as secret information pseudo-random function's (PRF) keys k_0, k_1 and a master secret key msk (a secret key for a strong signature scheme). The public interface of the hardware tokens is represented by the master public key mpk for msk. We refer to the party $p_n \in \mathcal{P}$ as the *leader*. Moreover, each execution of the protocol is uniquely identified by a session id sid $\in \mathbb{N}$.

A overview of the protocols in this chapter is shown in Fig. 4.1: (1) our simple CP protocol $\Pi^{\mathtt{HT}}$ that does not preserve security when tokens are compromised; (2) our CP protocol with fallback security for compromised tokens $\Pi^{\mathtt{HT\text{-}FBS}}$, which makes use of a secure MPC protocol and preserves its identifiable abort and fairness properties; (3) a penalisation scheme which disincentivises adversary abort by requiring deposit of collateral, for example on a smart-contract-enabled blockchain.

Collusion-preserving protocol via HT and non-aborting adversary with broadcast Roughly, our first protocol $\Pi^{\mathtt{HT}}$ works as follows: Each party $p_i \in \mathcal{P} - \{p_n\}$ sends his input, encrypted by his token $\mathtt{HT}_i$, to a designated leader party. Upon receipt, the leader gives these messages, along with his own input, to his own token. The token then computes the output, which the leader forwards to the other parties.

In more detail, each token $\mathtt{HT}_i$ uses $R_0 \leftarrow \mathsf{PRF}(k_0, \mathsf{sid})$ as randomness to generate an encryption key sk. In addition, it uses $R_1 \leftarrow \mathsf{PRF}(k_1, \mathsf{sid}||i)$[5] to generate a pair of session signing-verification $(\mathsf{sigk}_i, \mathsf{vk}_i)$ keys for a strong signature scheme and certifies them by signing $\mathsf{vk}_i||\mathsf{sid}||i$ with the master secret key msk thus obtaining cert_i.[6] We refer to the encryption key sk as the *session encryption key* and to $(\mathsf{sigk}_i, \mathsf{vk}_i)$ as the *session signing-verification keys*[7]. Note that the session encryption key sk is common to all the hardware tokens (since all the tokens share the same PRF keys).[8]

After the session keys have been generated, the hardware token $\mathtt{HT}_i$ encrypts his input x_i, signs this encrypted value together with f using sigk_i, and sends the encryption, the signature, and the certificate cert_i over the broadcast channel. The leader p_n collects all the encrypted values and signatures, and gives them to his hardware token $\mathtt{HT}_n$ along with his input x_n, the function f, and sid. His hardware token $\mathtt{HT}_n$ first checks that all certificates and signatures are valid and the inputs are consistent with the function f. Then, if all the checks are successful, $\mathtt{HT}_n$ generates, as described earlier, the session encryption key sk and decrypts all the encrypted inputs of the other parties using sk (we recall that the PRF key k_0 is shared among all the hardware tokens). Using everyone's (decrypted) inputs $x_1, \ldots, x_n$, , $\mathtt{HT}_n$ evaluates $y_1, \ldots, y_n = f(x_1, \ldots, x_n)$ and

[5] As an abuse of notation we refer to the identity of a party p_i with i.

[6] We use $||$ as the concatenation operator.

[7] Unless otherwise specified, a signing-verification key always refers to a *session* signing-verification key.

[8] Intuitively, we use session-keys instead of the master secret key because these keys will be leaked to the simulator. Leaking the master secret key would completely compromise the token. A more detailed discussion follows later this section.

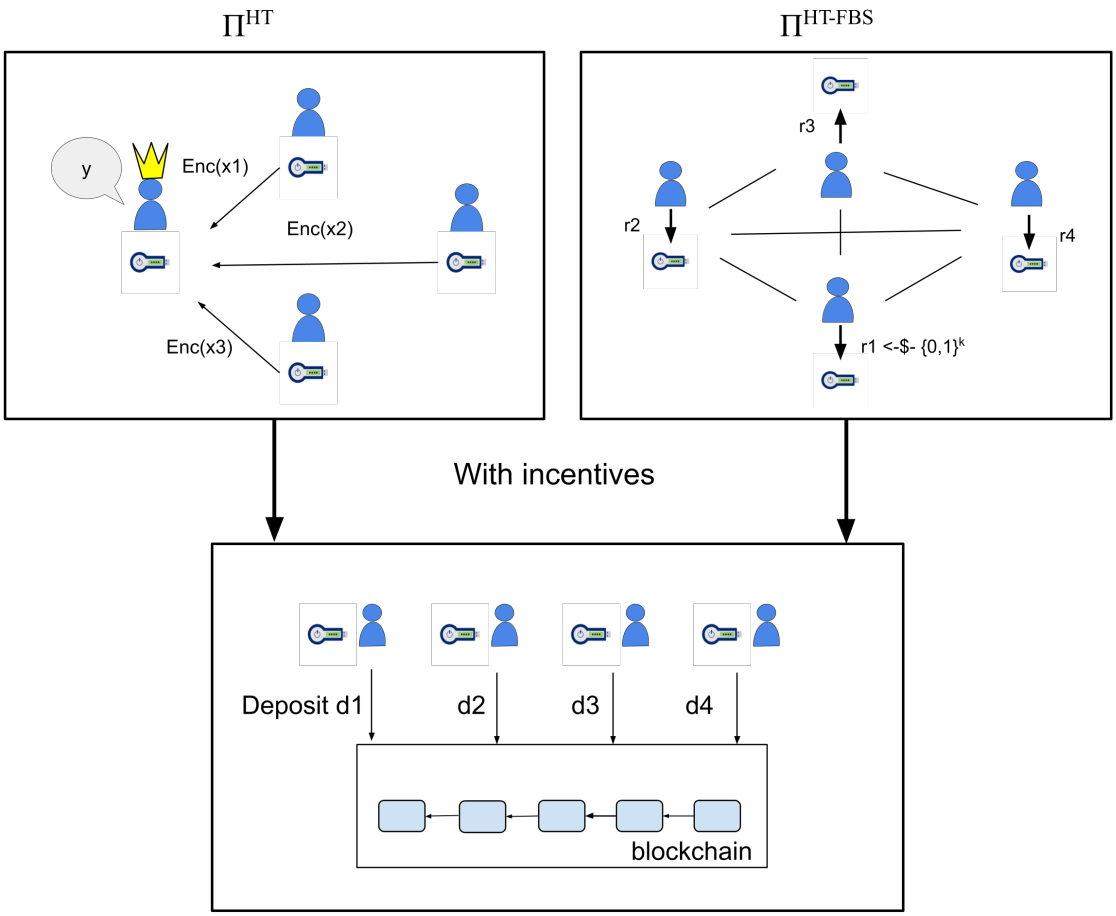

Figure 4.1: High level overview of the protocols in this chapter.

for $i = 1, \ldots, n-1$ encrypts y_i using sk and signs the (concatenation of the) encrypted values together with f using sigk_n. The encryptions and the signature uses randomness generated from evaluating $\text{PRF}(k_1, \text{sid}||n)$. Finally, the leader p_n propagates the output of $\texttt{HT}_n$ on the broadcast channel. Each party p_i, upon receiving a message, forwards it to $\texttt{HT}_i$, which verifies the certificate, the signature, and the consistency between f and sid. If the checks are successful then $\texttt{HT}_i$ uses sk to decrypt and output y_i.

Using hardware tokens allows us to achieve the following: (1) generate fresh randomness and session keys for each new session id sid, that are hidden from the parties themselves, and (2) certify that each sid is used only once. Intuitively, for (1), the randomness used in the computation must be hidden from the parties themselves to achieve CP over broadcast. A concrete example to demonstrate why hiding the randomness is important: Suppose a party Alice has access to a (limited) external channel and uses it to send this randomness to another party

Bob. If, for example, the randomness is used as Alice's decryption key in the protocol, Bob now can also decrypt messages directed towards Alice, since Bob sees these encrypted messages over broadcast. This breaks CP as the limited external communication led to additional information, e.g., Alice's output, to be leaked to Bob. For (2), to restrict any sid to one-time use, our stateful hardware token stops replying when a sid is used more than once. This is necessary to prevent an adversary from using the same sid (thus the same randomness) to evaluate different inputs. Otherwise, he can send a subliminal message by picking an input where, for example, the resulting encrypted message has its first two bits equal to the first two bits of his input—breaking collusion-preservation. This adversarial strategy was indeed observed in [100], limiting the games they consider to those with publicly observable actions (as informally explained in Sec. 4.1.2.1). The proof that that $\Pi^{\mathtt{HT}}$ is CP for non-aborting adversaries intuitively comes from the fact that $\Pi^{\mathtt{HT}}$ is deterministic given the tokens (which fixes the PRF and msk keys) and the sid we use. We note that while this may appear contradictory (i.e., to obtain a secure protocol you need entropy [76]), our protocol achieves security and collusion-preservation as the tokens generate fresh (pseudo)randomness for each sid (which is used only once). $\Pi^{\mathtt{HT}}$ also enjoys identifiable abort, and more interestingly, any external party observing the execution of the protocol without participating it can identify malicious behavior, by verifying signatures of messages on the channel. Following [93] we refer to this property as *publicly identifiable abort*. To reduce the amount of token memory required for checking that each sid is used only once, we propose two alternate solutions in Section 4.2.2.

Tokens in collusion-preserving computation One may wonder why hardware tokens cannot directly use their master secret key to authenticate protocol messages. The reason lies in the locality restrictions that the CP model places on the ideal world adversary (the simulator). Specifically, CP requires the existence of a local simulator $\mathcal{S}_i$ for each adversarial party p_i, and mandates that simulators cannot communicate with each other except if the environment explicitly allows them to. Thus, setups (in our case, tokens) which naturally introduce correlations between parties are tricky to define and use, especially when the protocol is executed over a broadcast channel. To understand the issue, one needs to observe that in a protocol over a broadcast channel, all parties expect to see exactly the same messages from this channel. Indeed, one of the novelties of our results is to show how in the real-world, i.e., in the protocol execution, the correlations embedded in the tokens can be leveraged to ensure that every (honest-protocol) broadcast message is predictable by any token. However, this correlation in the views inherently requires the simulators' views to be correlated in some way, in order to generate the same exact protocol messages. For instance, if $\mathcal{S}_1$ (the ideal world adversary with 1 as their ID) reports to the environment that he broadcasted message m in round ρ, then each $\mathcal{S}_j$ should also report to the environment that they heard this message. However, for this we need to allow the simulators to correlate their response. A naive first approach would be to allow them to interact over some underlying communication network. However, this defeats the purpose of collusion-preservation, as it explicitly introduces a venue of arbitrary correlations/collusion. A second approach would be to also offer the simulators access to correlated tokens. But this leads to a new technical issue: In order to simulate the token-hybrid protocol, our simulator needs to have extra control

of the hardware tokens (e.g., be able to program it). In fact, such asymmetry between the capabilities of the adversary and the simulator is proven necessary in various related settings (e.g., programmable random oracle).

We tackle the above issue by considering the hardware token as a (global) setup functionality and embedding a trapdoor inside. To ensure that only the simulators in the ideal world can use the trapdoor, we employ a technical trick inspired by [40] for the global random oracle. At a very high level, the trapdoor allows the simulators to produce the same signed messages, making their views on protocol messages consistent. Specifically, it gives access to a set of pre-computed messages which contain no information about the input of the parties. These messages are indistinguishable from the messages that would be generated in the real world and are properly signed with respect to the n signing-verification session keys as they would be in a real world execution. For example, for the protocol we have just described, the trapdoor would allow the simulator to get a set of encryptions of 0, all authenticated with respect to the corresponding signing-verification session keys. To enable such a mechanism, we introduce a token global-functionality that allows functionalities registered to it to send a special command $(\text{Trapdoor}, \text{sid})$. The registered functionalities can then relay the trapdoor information it receives to its simulators, allowing them to complete their simulations.

Most importantly, this trapdoor information remains useless for other protocols, preserving composability with other CP protocols. More concretely, consider an extension $\mathcal{F}^\star$ of $\mathcal{F}$, which behaves exactly as $\mathcal{F}$ but accepts an additional command GetTrapdoor from the ideal world adversary. In the simulation each simulator can send to $\mathcal{F}^\star$ the command $(\text{GetTrapdoor}, \text{sid})$. Upon receiving this command, $\mathcal{F}^\star$ sends $(\text{Trapdoor}, \text{sid})$ to the token functionality if and only if sid is equal its session id. The token functionality, upon receiving the command $(\text{Trapdoor}, \text{sid})$ from $\mathcal{F}^\star$, sends to $\mathcal{F}^\star$ the trapdoor information (e.g., the authenticated encryptions of 0). When $\mathcal{F}^\star$ receives a reply from the token functionality, it is forwarded to the simulator. Note that we leak only messages that can be used within a specific session id, since they are signed using signing key that are bonded to one session id. Indeed, as discussed previously, for each session we create new session (e.g., signature) keys that are valid only within that specific session.

The mechanism described above is quite natural as in the real world the parties are allowed to see signed messages passing on the channel and determine their actions based on these messages. The same should be allowed in the ideal world. Hence, we enhance the ideal functionality $\mathcal{F}$ by constructing $\mathcal{F}^\star$ which acts exactly as $\mathcal{F}$ as described above.

Collusion-preservation with fallback security Despite being simple and optimal in terms of round complexity, the protocol Π^{HT} above suffers from a big limitation. That is, if the hardware tokens are corrupted (e.g., the secret keys are leaked by the token manufacturer) then not only the CP-property is lost, but we cannot even guarantee to protect the honest parties' inputs. To rectify this issue we propose the protocol $\Pi^{\text{HT-FBS}}$ ("FBS" stands for fallback secure) that protects the input of the honest parties (in a standard GUC-security sense) even in the case where: (1) the adversary knows all the secret keys of the hardware tokens (including those held by honest parties) and (2) the malicious parties can arbitrarily modify or replace their own hardware tokens.

This protocol achieves a similar fallback security as the original collusion-preserving protocol

in the mediated model: When tokens are not compromised—and aborts are either excluded or deterred by means of incentives (see below)—then the protocol is collusion-preserving; and in any case (i.e., even when tokens are compromised) the protocol remains (G)UC secure—i.e., any profile of adversaries can be simulated by a monolithic simulator. We refer to a protocol that has such security guarantees as a *fallback secure* protocol.

Our fallback protocol guarantees also identifiable abort and unanimous abort for functionalities that guarantee termination, even when tokens can be broken. Interestingly, these properties are impossible to achieve in the analogous scenario of a corrupted mediator in the mediated model. To obtain such a protocol we use as a main building block a protocol Π^{MPC} that is secure against a malicious adversary and which enables identifiable (unanimous) abort. Each token computes protocol messages on behalf of its owner. In addition, the randomness used is jointly decided by the owner of the token and by the token itself.

More precisely, it is the XOR of the randomness produced by the hardware token, and a random string given at run-time by the token's owner. Intuitively, this means even if an honest party's token leaks its secret keys, it will still use an honestly-generated random string in the protocol. Thus, the party's input is protected by the security of the underlying MPC protocol, even if a token's secret keys are leaked. On the other hand, if all hardware tokens are uncompromised, then the randomness of any party in the MPC protocol becomes unknown and untamperable to everyone. Thus, no malicious party can send subliminal messages without being detected and causing an abort.

For sake of completeness, we also sketch a protocol which, although less round-efficient and without identifiable abort, preserves standard (GUC) security even when tokens are *fully* compromised—that is, not trusted, isolated, nor tamper-resilient. This is a stronger version of a "compromised" token since the adversary may read the contents or even change the behavior of honest parties' tokens. We make a simple alteration to our solution: Any computation performed by the token, will instead be done via a secure two-party protocol between the token and its owner. In more detail, each party p_j runs a secure 2-party protocol with his token, to obtain the next message of the protocol Π^{MPC}. When tokens are uncompromised, the behavior of this solution is the same as in our original solution, achieving CP. When tokens are compromised, fallback security follows from the security of 2-party protocol which ensures the tokens learn nothing. This solution, however, cannot achieve identifiable abort against fully-compromised tokens since any token can, e.g., be programmed to be unresponsive to any interactions. This also means the solution can only implement functionalities with abort, even with honest majority.

How to deal with aborting adversaries We note that the above CP-protocols cannot prevent, for example, a corrupt poker player from simply sending an encrypted message "*I have an ace of spades; let's collude!*" over the broadcast channel. While the protocol could detect such invalid messages and abort, this attack already breaks CP and seems unavoidable if no assumptions are made on the network topology (e.g., the mediated model [8, 9, 11]) and on the honesty of the network nodes. In this chapter, we show how to circumvent the above issue by considering an *incentive-driven* (rational) attacker. That is, we define a security notion called CPAP (collusion-preserving attack-payoff secure) that captures the fact that some adversarial actions—in our case

aborts—are not "for free" and instead incur a negative payoff. For example, as our protocols have (publicly) identifiable abort, a judge in the poker game example can identify and penalize an adversarial party causing the abort. Similar to the rational protocol design (RPD) framework [65], CPAP considers a CP-well-formed (Def. 25) functionality $\mathcal{F}$, and a relaxed functionality $\langle\mathcal{F}\rangle$ that acts the same as $\mathcal{F}$ except it explicitly includes weaknesses that allow a simulator to collude or abort. Then, we define a value function v mapping the joint view of a set of simulators[9] interacting with the relaxed functionality $\langle\mathcal{F}\rangle$ and the environment $\mathcal{Z}$, to a real-valued *payoff*. Intuitively, the real utility gained/lost by a set of adversaries for a given protocol is the payoff maximized over all environments, and minimized over all sets of simulators that successfully emulate the adversaries in the environment. For our CP protocols, we show that collusion always causes the real world execution to abort (and identify a corrupt party). Thus, when the cost of abort exceeds the gains from colluding, a rational attacker will never collude (otherwise simulator can trigger an abort in $\langle\mathcal{F}\rangle$)—intuitively it means that our protocol implements $\mathcal{F}$ for rational attackers.

From "ideal" to "real" payoffs. In Sec. 4.4 we construct a penalization protocol Π_{pen} which runs the computation of our CP protocols $\Pi^{\mathtt{HT}}$ or $\Pi^{\mathtt{HT\text{-}FBS}}$, and in addition penalizes the adversary when he colludes (which can be done only by triggering an abort). For a natural class of utilities for the protocol designer and attacker, Π_{pen} enjoys both CPAP and CPIC, following similar arguments as in Sec. 4.3.3.

4.2 Collusion-preserving (CP) MPC with non-aborting adversaries

In this section we present our protocol $\Pi^{\mathtt{HT}}$ for any CP-well-formed functionality (Def. 25) under the following assumptions: (1) each party has access to a hardware token (which we describe as one global ideal functionality $T^{\mathtt{HT}}$) (2) all communication is done over authenticated broadcast, and (3) adversarial parties do not make the protocol abort. For simplicity we restrict ourselves to non-reactive functionalities, also known as secure function evaluation. (The general case can be reduced to this case using a suitable form of secret sharing to maintain the secret intermediate states of the reactive functionality.) Moreover, we describe all our protocols in a round based, synchronous manner, where messages sent in some round are delivered by the beginning of the next round. We first introduce some additional notation:

- sid $\in \mathbb{N}$ uniquely identifies an execution of $\Pi^{\mathtt{HT}}$.
- Set of parties $\mathcal{P} = \{p_1, \ldots, p_n\}$ running $\Pi^{\mathtt{HT}}$ compute the function f.
- We call *leader* the party that is in charge to run a special code and we assume w.l.o.g. that the leader is $p_n \in \mathcal{P}$.
- $T^{\mathtt{HT}}$ denotes the global token functionality.

[9] Recall that in CP, adversaries are not monolithic, so we consider a set of adversaries/simulators instead of one single adversary/simulator.

For our construction we use the following tools:

- Pseudo-random functions: $\mathsf{PRF}_0 : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$, $\mathsf{PRF}_1 : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to \{0,1\}^{4\kappa}$, $\mathsf{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to \{0,1\}^{(n+2)\kappa}$, $\mathsf{PRF}_3 : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{(4n-2)\kappa}$;

- A strong unforgeable signature scheme

$$\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver});$$

- A secret-key encryption scheme

$$\Pi^{\mathsf{SK}} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}).$$

The global setup $\bar{\mathcal{G}}$ is represented by the token functionality $T^{\mathtt{HT}}$. We note that we use one functionality to *emulate* the behavior of the hardware tokens held by the parties that run the protocol. This token functionality replies to each party $p_i \in \mathcal{P}$ using the appropriate code and keys depending to the identity of the calling party (i.e. the functionality discriminates between leader and non-leader parties). To not overburden the notation, in the formal construction we denote the identity of a party $p_i \in \mathcal{P}$ with i. Moreover, the token functionality exports as public information the master public key mpk, and keep as part of its secret state the master secret key msk together with with the PRF keys K_0, K_1, K_2, K_3. The parties are allowed to communicate only via a broadcast channel denoted by $\mathcal{B}$ (c.f. Fig. 2.2 for formal definition).

We provide a formal description of $T^{\mathtt{HT}}$ in Fig. 4.2 and Fig. 4.3. The complete formal description of the the protocol $\Pi^{\mathtt{HT}}$ for the non-leader party is proposed in Fig. 4.4, and the protocol run by the leader parties is provided in Fig. 4.5. We assume that the ideal functionality $\mathcal{F}$ that we wish to realize is registered to the token functionality. In addition, upon receiving the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$, $\mathcal{F}$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to the token functionality if sid is equal its session id, and forwards the answer to the ideal adversary. We recall that this trapdoor allows us to capture the broadcast channel, on which all parties see the exact same signed messages. Equipping the ideal functionality with the trapdoor command translates this *real-world leakage* in the ideal world. We also recall that the functionality leaks to the simulators messages that are valid within one specific session without harming the token functionality globally.

We now prove that $\Pi^{\mathtt{HT}}$ is collusion-preserving against non-aborting adversaries for well-formed functionalities (Def. 25). Formally, we prove the following:

Theorem 5. *Let $\bar{\mathcal{G}} = T^{\mathtt{HT}}$ be the setup as defined above, $\mathcal{R} = \mathcal{B}$ (broadcast) and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality. Then the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\mathtt{HT}}$ (described by Fig. 4.4 and 4.5) CP realizes $\mathcal{F}$ in the $\mathcal{R}$-hybrid world assuming non-aborting adversaries.*

The token functionality is parameterized by a set of parties $\mathcal{P}$ and by a list $\overline{\mathcal{F}}$ of ideal functionality programs. The functionality manages the keys $(\mathsf{mpk}, \mathsf{msk})$ for the signature scheme Σ and the PRF keys K_0, K_1, K_2, K_3.
If $I = (\mathsf{Get\_key}, \mathsf{sid})$ is received return mpk to the caller.

Input phase for non-leader parties.
If $I = (\mathsf{Input}, \mathsf{sid}, x, f)$ is received from a non-leader party p_j then do the following.

- If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
- Compute $R_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid})$ and $\mathsf{Kenc}^{\mathsf{sid}} \leftarrow \mathsf{Gen}(1^\kappa; R_0)$
- Compute $R_1 \leftarrow \mathsf{PRF}_1(K_1, \mathsf{sid}||j)$ and parse R_1 as 4 strings of κ bits each $\mathsf{rs}^1||\mathsf{rs}^2||r^1||r^2$.
- $(\mathsf{sigk}_j^{\mathsf{sid}}, \mathsf{vk}_j^{\mathsf{sid}}) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}^1)$
- $\mathsf{cert}_j \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j^{\mathsf{sid}}||\mathsf{sid}||j; \mathsf{rs}^2)$
- Compute $\overline{x}, \sigma \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, x; r^1), \mathsf{Sign}(\mathsf{sigk}_j^{\mathsf{sid}}, \overline{x}||f; r^2)$ and output $(\overline{x}, f, \mathsf{vk}_j, \sigma, \mathsf{cert}_j)$.

Output phase for non-leader parties. If $I = (\mathsf{Output}, \mathsf{sid}, z)$ is received, parse z as $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ and do the following. If $\mathsf{Ver}(\mathsf{vk}_n, \overline{y}_1|| \ldots ||\overline{y}_{n-1}||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_j||\mathsf{sid}||j, \mathsf{cert}_j) = 1$ then compute and output $\mathsf{Dec}(\mathsf{Kenc}^{\mathsf{sid}}, \overline{y}_j)$, output $\perp$ otherwise.

Trapdoor. If $I = (\mathsf{Trapdoor}, \mathsf{sid})$ is received from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$ then do the following.

- If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
- Pick $r_1|| \ldots ||r_{n-1}||r'_1|| \ldots ||r'_{n-1}||\mathsf{rs}_1^1||\mathsf{rs}_1^2|| \ldots ||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_3(K_3, \mathsf{sid})$.
- For each $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$. For each $j \in [n-1]$ $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}_n, 0^\kappa; r_j)$, $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, e_j; r'_j)$
- For each $j \in [n-1]$ compute $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, 0^\kappa; r'_j)$
- $\sigma_n \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, \overline{y}_1|| \ldots ||\overline{y}_{n-1}||f||\mathsf{sid})$.
- Return to the calling instance $\{\mathsf{vk}_i, \mathsf{cert}_i, \sigma_i\}_{i\in[n]}, \{e_i, \overline{y}_i\}_{i\in[n-1]}$.

Figure 4.2: Functionality $T^{\mathtt{HT}}$ for the behavior of hardware tokens of non-leader parties.

Proof sketch. For simplicity we assume that only the leader is honest. To prove this theorem, we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ that satisfy the CP definition, Definition 11. For $i = 1, \ldots n$, the simulator $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A}_i)$ queries $(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If so, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates a set of encryptions of 0^κ, a set of signing/verification keys and uses them to authenticate these encryptions (see the bottom of Fig. 4.2/Fig. 4.3). We note that each simulator will obtain the same set of ciphertexts and verification keys. This is crucial for the proof to go through, as each individual simulator $\mathcal{S}_i$ internally runs the corrupted party p_i, and it will act on the behalf of the other $n-1$ parties using the authenticated cipthertexts received by $\mathcal{F}^\star$. $\mathcal{S}_i$ will also intercept all the queries the party p_i makes to $T^{\mathtt{HT}}$. Assuming that p_i is non-aborting, then at some point they will query $T^{\mathtt{HT}}$ with their input x_i. $\mathcal{S}_i$ now has the input of the corrupted party, and he will use it to query

The token functionality is parameterized by a set of parties $\mathcal{P}$ and by a list $\overline{\mathcal{F}}$ of ideal functionality programs. The functionality manages the keys $(\mathsf{mpk}, \mathsf{msk})$ for the signature scheme Σ and the PRF keys K_0, K_1, K_2, K_3.
If $I = (\mathsf{Get\_key}, \mathsf{sid})$ is received return mpk to the caller.

Input/output phase for the leader party.
If $I = (\mathsf{Input}, \mathsf{sid}, x_n, f, \mathsf{sid}, (\overline{x}_1, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \ldots, (\overline{x}_{n-1}, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1}))$ is received from the leader party p_n then check if for all $j \in [n-1]$ $\mathsf{Ver}(\mathsf{vk}_j, \overline{x}_j||f, \sigma_j) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_j||\mathsf{sid}||j, \mathsf{cert}_j) = 1$. If it is not, then output $\perp$ and stop, otherwise act as follows.

- If $\mathsf{ctr}_n^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_n^{\mathsf{sid}} \leftarrow 1$ else output $\perp$ and stop.
- $R_2 \leftarrow \mathsf{PRF}_2(K_2, \mathsf{sid}||n)$; parse as $n+2$ strings of κ bits each $\mathsf{rs}^1||\mathsf{rs}^2||r_1||r_2||\ldots||r_{n-1}||r^\star$.
- $(\mathsf{sigk}_n^{\mathsf{sid}}, \mathsf{vk}_n^{\mathsf{sid}}) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}^1)$
- $\mathsf{cert}_n \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_n^{\mathsf{sid}}||\mathsf{sid}||n; \mathsf{rs}^2)$
- Compute $R_0 \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid})$ and $\mathsf{Kenc}^{\mathsf{sid}} \leftarrow \mathsf{Gen}(1^\kappa; R_0)$.
- For $j = 1, \ldots, n-1$ compute $x_j \leftarrow \mathsf{Dec}(\mathsf{Kenc}^{\mathsf{sid}}, \overline{x}_j)$.
- Compute $y_1, \ldots, y_n \leftarrow f(x_1, \ldots, x_n)$.
- For $j = 1, \ldots, n-1$ compute $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{Kenc}^{\mathsf{sid}}, y_j; r_j)$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_n^{\mathsf{sid}}, \overline{y}_1||\ldots||\overline{y}_{n-1}||f; r^\star)$;
- Output $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n), y_n$

Trapdoor. If $I = (\mathsf{Trapdoor}, \mathsf{sid})$ is received from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$ then do the following.

- If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
- Pick $r_1||\ldots||r_{n-1}||r'_1||\ldots||r'_{n-1}||\mathsf{rs}_1^1||\mathsf{rs}_1^2||\ldots||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_3(K_3, \mathsf{sid})$.
- For each $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$. For each $j \in [n-1]$ $e_j \leftarrow \mathsf{Enc}(\mathsf{pk}_n, 0^\kappa; r_j)$, $\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, e_j; r'_j)$
- For each $j \in [n-1]$ compute $\overline{y}_j \leftarrow \mathsf{Enc}(\mathsf{pk}_j, 0^\kappa; r'_j)$
- $\sigma_n \leftarrow \mathsf{Sign}(\mathsf{sigk}_n, \overline{y}_1||\ldots||\overline{y}_{n-1}||f||\mathsf{sid})$.
- Return to the calling instance $\{\mathsf{vk}_i, \mathsf{cert}_i, \sigma_i\}_{i\in[n]}, \{e_i, \overline{y}_i\}_{i\in[n-1]}$.

Figure 4.3: Functionality $T^{\mathtt{HT}}$ for the behavior of hardware tokens for the leader party.

the ideal functionality. In addition, $\mathcal{S}_i$ sends to p_i (acting on the behalf of $T^{\mathtt{HT}}$) an encryption of 0 authenticated with respect to the verification key vk_i. Assuming that no party aborts, the simulation is successful since the messages generated by the individual simulator on the locally simulated broadcast channels are exactly the same for all the parties (unless the adversary breaks the signature scheme).

Full proof. To prove CP, we must show n independent simulators $\mathcal{S}_1, \ldots, \mathcal{S}_n$. At a very high level for all $i \in \{1, \ldots, n\}$, $\mathcal{S}_i$ can query $\mathcal{F}$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}$ checks that sid is equal to its session id, and if so, it sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates the string

We assume that the party p_j is registered to the token functionality $T^{\mathtt{HT}}$ and that it obtains mpk by querying it with $I = (\mathsf{Get\_key}, \mathsf{sid})$. Each party is aware of the function that will be computed f, of the identifier of each execution sid, and of the parties involved in each of those executions $\mathcal{P}$.

Input.

- The party p_j on input $(\mathsf{Compute}, \mathsf{sid}, x)$ sends $(\mathsf{Input}, \mathsf{sid}, x, f)$ to $T^{\mathtt{HT}}$.
- Upon receiving the answer X from $T^{\mathtt{HT}}$, if $X = \perp$ then p_j outputs $\perp$ and stops. Otherwise, p_j sends X to p_n.

Output. The party p_j, upon receiving $z = (\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ from p_n sends $(\mathsf{Output}, \mathsf{sid}, z)$ to $T^{\mathtt{HT}}$. Upon receiving y from $T^{\mathtt{HT}}$, p_j outputs y.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, f', \mathsf{vk}, \sigma, \mathsf{cert})$ is received from a party p_i check if $f = f'$ and $\mathsf{Ver}(\mathsf{vk}_i, m||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}||\mathsf{sid}||i, \mathsf{cert}) = 1$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 4.4: Protocol executed by the party p_j.

Input/output

- The party p_n on input $(\mathsf{Compute}, \mathsf{sid}, x_n)$ collects messages from $p_{j\in[n-1]}$ and sends $I = (\mathsf{Input}, x_n, f, \mathsf{sid}, (\overline{x}_1, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \ldots, (\overline{x}_{n-1}, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1}))$ to $T^{\mathtt{HT}}$.
- Upon receiving the answer Y from $T^{\mathtt{HT}}$, if $Y = \perp$ then output $\perp$ and stop. Otherwise parse Y as $((m, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n), y)$, send $(m, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to all the parties in $\mathcal{P}$ and output y.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, f', \mathsf{vk}, \sigma, \mathsf{cert})$ is received from a party p_i check if $f = f'$ and $\mathsf{Ver}(\mathsf{vk}_j, m||f, \sigma) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}||\mathsf{sid}||j, \mathsf{cert}) = 1$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 4.5: Protocol executed by the leader party p_n.

$\widetilde{K}$, n pairs of signing-verification keys, and authenticates the verification keys using the master secret key msk. $T^{\mathtt{HT}}$ then sends this information to $\mathcal{F}$ which forwards them to the simulator $\mathcal{S}_i$. Note that if all the simulators query $\mathcal{F}$ with $(\mathsf{GetTrapdoor}, \mathsf{sid})$ they will all get the same pair of authenticated signing-verification key and the string $\widetilde{K}$. Given the above information, a simulator $\mathcal{S}_i$ can use $\widetilde{K}$ as input of a PRG to generate the randomness sufficient to: 1) to create a key $\mathsf{Kenc}^{\mathsf{sid}}$ for a secret-key encryption scheme 2) compute $n+1$ encryptions of 0^κ and 3) authenticate the encrypted value using the appropriate signature session keys. These authenticated messages are now used to interact with $p_i^\star$ (who is not supposed to distinguish between the encryptions of 0^κ and the encryptions that contain the actual inputs of the parties). Moreover, whenever $\mathcal{S}_i$ receives the input form $p_i^\star$ he forwards it to the $\mathcal{F}$ which returns the output y_i. The crucial observation is that the messages seen by all the corrupted parties are the same since the simulators use exactly the same strategy and randomness, and since the adversary cannot forge a signature for the strong signature scheme we are using[10]. We now provide a more formal argument for the proof.

We start by assuming that at least one party is honest. In order to prove this part of the theorem we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\mathtt{HT}},\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$$

For $i = 1, \ldots n$, the simulator $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A}_i)$ queries$(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If it is, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\mathtt{HT}}$. $T^{\mathtt{HT}}$ then generates the string $\widetilde{K}$, n couples of signing-verification keys, and authenticates the verification keys using the master secret key msk (as shown in Fig 4.2/Fig. 4.3) thus obtaining $\{\mathsf{vk}_i, \mathsf{cert}_i, \sigma_i\}_{i\in[n]}, \{e_i, \overline{y}_i\}_{i\in[n-1]}$. $\mathcal{F}^\star$ then sends $T^{\mathtt{HT}}$ this information to $\mathcal{F}^\star$ which forwards them to the simulator $\mathcal{S}_i$.

From here onwards the behaviors of the simulators differ. Without loss of generality we describe how the simulator $\mathcal{S}_1$ works since $\mathcal{S}_i$ with $i \in \{2, \ldots, n-1\}$ will follow exactly the same strategy as $\mathcal{S}_1$. We then show how the simulator $\mathcal{S}_n$ works.

$\mathcal{S}_1$: The simulator $\mathcal{S}_1$ internally runs the adversary $\mathcal{A}_1$, emulates the token functionality $T^{\mathtt{HT}}$ for $\mathcal{A}_1$ for the session id sid and acts on the behalf of the parties $p_2, \ldots, p_n$. $\mathcal{S}_1$ executes the following steps.

1. Send $(e_2, f, \mathsf{vk}_2, \sigma_2, \mathsf{cert}_2), \ldots, (e_{n-1}, f, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1})$ to $\mathcal{A}_1$.
2. If $\mathcal{A}_1$ sends $I = (\mathsf{Input}, \mathsf{sid}, x, f')$ to $T^{\mathtt{HT}}$ then do the following.
   - If $\mathsf{ctr}_1^{\mathsf{sid}}$ is not defined then define it and set $\mathsf{ctr}_1^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
   - Output $(e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1)$

[10] We note that it is crucial to use a strong signature scheme to avoid the creation of a different valid signature for a message m from valid signatures for the same message.

3. If $\mathcal{A}_1$ sends $(e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1)$ over broadcast then send (sid, x_1) to $\mathcal{F}$.
4. Upon receiving y_1 from $\mathcal{F}$ send $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to $\mathcal{A}_1$.
5. Upon receiving $(\mathsf{Output}, \mathsf{sid}, z)$ from p_1, if $z = (\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ then send y_1 to $\mathcal{A}_1$, output $\perp$ otherwise.

$\mathcal{S}_n$**:** The simulator $\mathcal{S}_n$ internally runs the adversary $p_n^\star$, emulates the token functionality $T^{\mathtt{HT}}$ for $p_n^\star$ for the session id sid and acts on the behalf of the parties $p_1, \ldots, p_{n-1}$. $\mathcal{S}_n$ executes the following steps.

1. Send $((e_1, f, \mathsf{vk}_1, \sigma_1, \mathsf{cert}_1), \ldots, (e_{n-1}, f, \mathsf{vk}_{n-1}, \sigma_{n-1}, \mathsf{cert}_{n-1}))$ to $p_n^\star$.
2. If $I = (\mathsf{Input}, \mathsf{sid}, x_n, f', (e'_1, f_1, \mathsf{vk}'_1, \sigma'_1, \mathsf{cert}'_1), \ldots, (e'_{n-1}, f_{n-1}, \mathsf{vk}_{n-1}, \sigma'_{n-1}, \mathsf{cert}'_{n-1}))$ is received from $p_n^\star$, then check if for $j = 1, \ldots, n-1$: $e'_j = e_j$, $\sigma'_j = \sigma_j$, $\mathsf{vk}_j = \mathsf{vk}'_j$ and $\mathsf{cert}_i = \mathsf{cert}'_i$. If it is not, then output $\perp$ and stop, otherwise execute the following steps.
   - If $\mathsf{ctr}_n^{\mathsf{sid}}$ is not defined then defined it and set $\mathsf{ctr}_n^{\mathsf{sid}} \leftarrow 1$ otherwise output $\perp$ and stop.
   - Send x_n to $\mathcal{F}$ and upon receiving y_n from $\mathcal{F}$ send $(\overline{y}_1, \ldots, \overline{y}_{n-1}, f, \mathsf{vk}_n, \sigma, \mathsf{cert}_n)$ to $p_n^\star$.

The main differences from the real world are that the adversarial parties see dummy encryptions instead of the encryptions. We note that the simulation strongly relies on the fact that the adversaries cannot forge the signatures output of the hardware token functionalities. Indeed, by forging a signature an adversary could: (1) change the input of another party, or (2) use the inputs of the honest parties to evaluate a functions $f' \neq f$ or (3) evaluate the same function multiple times on the same honest parties inputs by changing the value sid thus completely breaking the security of the protocol.

In the case that all the parties are corrupted then we rely on the fact that $\mathcal{F}$ is CP-well-formed functionality and we allow the adversarial parties to communicate freely via $\mathcal{F}$. More precisely,

1. Whenever a message m is received on the ith adversarial interface, $\mathcal{F}$ outputs (i, m) to the first adversarial interface.
2. Whenever a message of the form (i, m) is received on the first adversarial interface, $\mathcal{F}$ outputs the message m to the ith adversarial interface.

□

4.2.1 (Publicly) Identifiable abort

Another interesting property enjoyed by $\Pi^{\mathtt{HT}}$ is *identifiable abort*. Recall from the preliminaries (Section 2.4.1.3) that a protocol run by a set of parties $\mathcal{P}$ is said to be *secure with identifiable abort* if it either computes according to its specification, or it aborts with the index of some corrupted party $p_i \in \mathcal{P}$ —i.e., every honest party learns the identity of a corrupted p_i.

In $\Pi^{\mathtt{HT}}$ the adversary can only deviate from the protocol specification by either sending a message authenticated with respect to a sid′ or f' not equal to the correct sid or f the honest parties use, sending a message with an invalid signature or certificate, or fail to send a message. Each event is verifiable by honest parties, and even third parties not involved in the protocol. Indeed, with the master public key mpk, sid and function f, it is possible to claim who did abort in a run of $\Pi^{\mathtt{HT}}$ by just inspecting its transcript. Formally, the protocol $\Pi^{\mathtt{HT}}$ securely realizes the function $\mathcal{F}_{\mathsf{IDA}}^{f}$, where $\mathcal{F}_{\mathsf{IDA}}^{f}$ involves n parties. More interestingly, we can modify $\Pi^{\mathtt{HT}}$ to support an additional party p_{n+1} which takes no input, does not send any message and outputs a default value (e.g., 0). Since p_{n+1} knows the master public key mpk, she can check the validity of the signature and the certificate. Hence, she is able to identify an invalid message (in the case p_{n+1} is honest). That is, our protocol allows an *observer* of the protocol execution to identify a misbehaving party. Following [93] we refer to this property as *publicly identifiable abort*, and to p_{n+1} as a *judge*. The code of the judge can be used by anyone who has the public setup and wants to follow the protocol execution and decide who aborted the protocol given the parties' messages.

4.2.2 Note on implementing $T^{\mathtt{HT}}$ with real hardware tokens

Following the approach of [10], we describe the behavior of the hardware tokens (HTs) by means of a single ideal functionality with a single set of shared keys. In particular, the master signing/verification keys allow tokens and parties to verify whether a message was signed by a hardware token. In practice, such functionality can be replaced by tokens that each has its own secret information (e.g., the HTs do not need share the same master signing key). More precisely, only a global master verification key needs to be shared among the tokens. Each hardware token manages its own signing and verification keys, msk_i and mpk_i, along with a certificate c_i which is a signature of mpk_i created by the HT manufacturer's global master secret key. An example of this approach is Intel SGX processors, where each processor has a unique attestation key and "endorsement certificate" from the manufacturer [50]. To authenticate a message, HT HT_i signs it with its msk_i, and sends the signature together with his master verification key vk_i and the certificate c_i. Anybody that has the global master verification key can verify that the certificate c_i is valid for vk_i, and that the signature issued by HT_i is valid w.r.t. vk_i.

Another important part of $T^{\mathtt{HT}}$ is to ensure that the session ID sid must not be reused in different protocol sessions. The most simple solution stores all sids that the token has seen. Thus, if the token cannot store more sids, it will not be able to verify the freshness of new sids and must stop responding all together. This in effect makes the token no longer usable for our CP protocol. We present two alterations to improve upon the space usage of the simple solution. First, the token can transfer the burden of storing the sids to an external memory, in a hash chain data structure [97]. The token stores the head and tail of the hash chain, which ensures that no malicious party can tamper with the sids on the external memory. The solution only requires a small (i.e., constant in the number of sids) amount of storage. However, verifying a sid requires interaction with the external memory to retrieve the hash chain—to use minimal space, the token may choose to retrieve the chain one hash at a time. To eliminate interactions with external memory, one may also opt for a Bloom filter [29]. This solution trades off the need for

interaction with the possibility (depending on the space allocated to the filter) of falsely marking some sids as "used". This however does not impact security, as it is equivalent to the token having seen a session with this sid.

4.2.3 Fallback solution when tokens can be compromised

While simple and optimal in terms of round complexity, the protocol Π^{HT} cannot guarantee any form of security when hardware tokens are corrupted/compromised. For example, if the secret keys are leaked by the token manufacturer then not only the CP-property is lost, but we cannot even guarantee any protection with regard to the inputs of the honest parties. In this section we propose a CP protocol $\Pi^{\text{HT-FBS}}$ that provides fallback security (in a standard GUC-security sense) in the following corruption model: (1) the secret keys of the tokens, including those of honest parties, can be leaked and (2) the memory/code of corrupt parties' tokens can be read/modified completely. Let $\mathcal{F}$ be the function that the parties wish to compute. For our construction we use the following tools.

- Pseudo-random functions $\mathsf{PRF}_0 : \{0,1\}^\kappa \times \{0,1\}^{2\kappa} \to \{0,1\}^{(2m+4)\kappa}$ and $\mathsf{PRF}_1 : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^{((m+3)n+1)\kappa}$
- A strong unforgeable signature scheme $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Ver})$.
- A n-party MPC protocol $\Pi^{\mathsf{MPC}} = (\mathsf{Next}_1, \ldots, \mathsf{Next}_n)$ that GUC-realizes functionality $\mathcal{F}$.

The global setup $\bar{\mathcal{G}}$ is represented by the token functionality $T^{\text{HT-FBS}}$. Similar to the token functionality of the previous section, $T^{\text{HT-FBS}}$ has a master public key mpk and a secret state consisting of the corresponding master secret key msk and the PRF keys K_0, K_1. We assume without loss of generality that the setup required to run Π^{MPC} is part of $T^{\text{HT-FBS}}$. We denote our protocol with $\Pi^{\text{HT-FBS}}$ (Fig. 4.7) and provide a formal description of $T^{\text{HT-FBS}}$ in Fig. 4.6.

Security of $\Pi^{\text{HT-FBS}}$:

We summarize the properties of the protocol $\Pi^{\text{HT-FBS}}$.

1. If the hardware tokens are not compromised, and no party aborts, then $\Pi^{\text{HT-FBS}}$ is collusion-preserving.
2. If the hardware tokens are compromised and Π^{MPC} GUC realizes $\mathcal{F}_{\mathsf{AB}}^f$ with $\mathsf{AB} \in \{\mathsf{IDA}, \mathsf{UNA}\}$, then $\Pi^{\text{HT-FBS}}$ GUC realizes $\mathcal{F}_{\mathsf{AB}}^f$
3. If the hardware tokens are not compromised (but the malicious parties may abort), then $\Pi^{\text{HT-FBS}}$ GUC realizes the functionality $\mathcal{F}$ with publicly identifiable abort.

The properties 1 and 2 above enable the fallback security of $\Pi^{\text{HT-FBS}}$. In addition, the second property states that in the case of corrupted tokens, $\Pi^{\text{HT-FBS}}$ inherits all the properties of Π^{MPC} (e.g., identifiable abort). We note that if the MPC protocol guarantees fairness (or even output delivery), this property would be held by $\Pi^{\text{HT-FBS}}$ as well. The third property states that if an adversarial party aborts then the CP property might be lost, but the input of the honest parties

are protected. We capture the case where the hardware tokens are compromised by considering the token functionality $\overline{T}^{\text{HT-FBS}}$ instead of $T^{\text{HT-FBS}}$. $\overline{T}^{\text{HT-FBS}}$ extends $T^{\text{HT-FBS}}$ with the additional command Tamper. If the adversary queries the token functionality with Tamper then $\overline{T}^{\text{HT-FBS}}$ leaks to the adversary its secret state (i.e., the master secret key msk and the PRF keys). Given the master secret key, the adversary can authenticate any message he wants and therefore acts on the behalf of the hardware token. To formally prove that $\Pi^{\text{HT-FBS}}$ is fallback secure we need to prove the following two lemmata.

Intuitively, to prove Lemma 9 we rely on the fact that any execution of $\Pi^{\text{HT-FBS}}$ can be seen as an execution of Π^{MPC} among honest parties. Indeed, in the case the hardware tokens are not corrupted the adversary has no control on the messages of Π^{MPC}, and he can only act as a proxy between the hardware tokens and the broadcast channel (since we assume that the adversary is non-aborting). This allows the CP simulators $\mathcal{S}_1, \ldots, \mathcal{S}_n$ to internally run the simulator Sim of Π^{MPC} (that exists by definition) for the case where no parties are corrupted. Hence, $\mathcal{S}_1, \ldots, \mathcal{S}_n$ can just run Sim using the same correlated randomness obtained from the trapdoor information $\widetilde{K}$, following the approach proposed in Sec 4.2. To prove Lemma 10 we can reduce the security of the entire protocol to the GUC security of Π^{MPC}. We note that in Lemma 10 the global setup is represented by $\overline{T}^{\text{HT-FBS}}$, which captures the scenario where the hardware tokens are compromised. We now provide more formal arguments.

Lemma 9. *Let $\bar{\mathcal{G}} = T^{\text{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$ (broadcast) and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality. Then the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\text{HT-FBS}}$ (described by Fig. 4.7) CP realizes $\mathcal{F}$ in the $\mathcal{R}$-hybrid world assuming that no parties abort.*

Proof. We start the proof by assuming that at least one party is honest (otherwise the lemma trivially holds since we assume a CP-well-formed functionality). In order to prove this part of the theorem we need to show a collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\text{HT-FBS}},\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$$

By assumption on the security of the MPC protocol, we know that $\forall\mathcal{A}\ \exists\mathsf{S}\ \forall\mathcal{Z}$ such that $\text{EXEC}^{\bar{\mathcal{G}},\mathcal{B}}_{\Pi^{\text{MPC}},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{F}}_{\mathsf{S},\mathcal{Z}}$.

We consider now S (the MPC simulator) for the case where there are no corrupted parties and describe how $\mathcal{S}_i = \mathsf{Sim}(\mathcal{A})_i$ works for $i = 1, \ldots n$. Without loss of generality we formally describe how the simulator $\mathcal{S}_1$ works since the other simulators follow exactly the same strategy.

The simulator $\mathcal{S}_1$ queries $(\mathsf{GetTrapdoor}, \mathsf{sid})$ $\mathcal{F}^\star$ with the command $(\mathsf{GetTrapdoor}, \mathsf{sid})$. $\mathcal{F}^\star$ checks that sid is equal to its session id. If it is, then $\mathcal{F}^\star$ sends $(\mathsf{Trapdoor}, \mathsf{sid})$ to $T^{\text{HT-FBS}}$.

$T^{\text{HT-FBS}}$ sends $\{\mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]}, \{\mathtt{msg}^\ell_j, \sigma^\ell_j\}_{j\in[n],\ell\in[m]}$ information to $\mathcal{F}^\star$ which forwards them to the simulator $\mathcal{S}_1$. Then $\mathcal{S}_1$ executes the following steps.

- Send $(\mathtt{msg}^\ell_2, \mathsf{vk}_2, \sigma^\ell_2, \mathsf{cert}_2), \ldots, (\mathtt{msg}^\ell_n, \mathsf{vk}_n, \sigma^\ell_n, \mathsf{cert}_n)$ to $\mathcal{A}_1$.

The functionality manages the keys $(\mathsf{mpk}, \mathsf{msk})$ for the signature scheme Σ. The functionality manages also the PRF keys K_0, K_1. If $I = (\mathsf{Get\_key}, \mathsf{sid})$ is received return mpk to the caller.

Input phase. If $I = (\mathsf{Input}, \mathsf{sid}, x_j, R_j)$ is received from some party p_j, then do the following,

- If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined, then define it and $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$, otherwise output $\perp$ and stop.
- Compute $R_0^{\mathsf{sid}} \leftarrow \mathsf{PRF}_0(K_0, \mathsf{sid}||j) \oplus R_j$ and parse R_0^{sid} as $(2m+4)$ strings of κ bits $\mathsf{rs}_j^1||\mathsf{rs}_j^2||\rho_j^1||\ldots||\rho_j^{m+1}||r_j^1||\ldots||r_j^{m+1}$.
- $(\mathsf{sigk}_j^{\mathsf{sid}}, \mathsf{vk}_j^{\mathsf{sid}}) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}_j^1)$
- $\mathsf{cert}_j^{\mathsf{sid}} \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_j||\mathsf{sid}||j; \mathsf{rs}_j^2)$
- Output the first message $(\mathtt{msg}_j^1, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\mathtt{msg}_j^1 = \mathsf{Next}_j(1^\kappa, x_j, \rho_j^1, \perp)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}_j^1||\Pi^{\mathsf{MPC}}||\ell; r_j^1)$

Next message function. If received $I = (\mathsf{NextMsg}, \mathsf{sid}, \{\mathtt{msg}_i^\ell, \sigma_i^\ell, \mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]})$ from p_j then:

- If $\ell > m$ or $\mathsf{ctrap}_j^{\mathsf{sid}} = 1$ then output $\perp$ and stop, else continue with the following steps.
- Parse R_0^{sid} as $\mathsf{rs}_j^1||\mathsf{rs}_j^2||\rho_j^1||\ldots||\rho_j^{m+1}||r_j^1||\ldots||r_j^{m+1}$.
- Store $\mathtt{msg}_{\mathsf{sid}}^\ell = \{\mathtt{msg}_i^\ell\}_{i\in[n]}$.
- For all $i \in [n]$ check if $\mathsf{Ver}(\mathsf{vk}_i, \mathtt{msg}_i^\ell||\Pi^{\mathsf{MPC}}||\ell, \sigma_i^\ell) = 1$ and $\mathsf{Ver}(\mathsf{mpk}, \mathsf{vk}_i||\mathsf{sid}||i, \mathsf{cert}_i) = 1$. If it is not then output $(p_i, \perp)$ and stop. Otherwise continue with the following steps.
- Set $\ell \leftarrow \ell + 1$, compute $\mathtt{msg}_j^\ell = \mathsf{Next}_j(1^\kappa, x_j, \rho_j^\ell, \overline{\mathtt{msg}}^{<\ell})$ with $\overline{\mathtt{msg}}^{<\ell} = \{\mathtt{msg}_i^{\ell'}\}_{i\in[n],\ell'<\ell}$ where $\mathtt{msg}_i^{\ell'}$ is the message from p_i at round ℓ' that was stored in $\mathtt{msg}_{\mathsf{sid}}^{\ell'}$
- If $\ell = m+1$ then output $y_j \leftarrow \mathtt{msg}_j^\ell$

  else, output $(\mathtt{msg}_j^\ell, \mathsf{sid}, \Pi^{\mathsf{MPC}}, \sigma_j^\ell, \mathsf{vk}_j, \mathsf{cert}_j)$, where $\sigma_j^\ell \leftarrow \mathsf{Sign}(\mathsf{sigk}_j^{\mathsf{sid}}, \mathtt{msg}_j^\ell||\Pi^{\mathsf{MPC}}||\ell; r_j^\ell)$.

Trapdoor. If received $I = (\mathsf{Trapdoor}, \mathsf{sid})$ from instance of functionality in the list $\overline{\mathcal{F}}$ then:

- If $\mathsf{ctr}_j^{\mathsf{sid}}$ is not defined, then define it, define $\mathsf{ctrap}_j^{\mathsf{sid}}$, set $\mathsf{ctr}_j^{\mathsf{sid}} \leftarrow 1$ and $\mathsf{ctrap}_j^{\mathsf{sid}} \leftarrow 1$. Else output $\perp$ and stop.
- Compute $\rho||r_1^1||\ldots||r_1^{m+1}||\ldots||r_n^1||\ldots||r_n^{m+1}||\mathsf{rs}_1^1||\mathsf{rs}_1^2||\ldots||\mathsf{rs}_n^1||\mathsf{rs}_n^2 \leftarrow \mathsf{PRF}_1(K_1, \mathsf{sid})$.
- For all $i \in [n]$ $(\mathsf{sigk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Kgen}(1^\kappa; \mathsf{rs}_i^1)$, $\mathsf{cert}_i \leftarrow \mathsf{Sign}(\mathsf{msk}, \mathsf{vk}_i||\mathsf{sid}||i; \mathsf{rs}_i^2)$.
- Use ρ to run the simulator of the MPC, S for the case where there are no corrupted party.
- Let $\{\mathtt{msg}_j^\ell\}_{j\in[n],\ell\in[m]}$ be the messages contained in the transcript obtained by the MPC simulator S. For all $j \in [n]$ and $\ell \in [m]$ computes $\sigma_j^\ell \leftarrow \mathsf{Sign}(\mathsf{sigk}_j, \mathtt{msg}_j^\ell||\Pi^{\mathsf{MPC}}||\ell; r_j^\ell)$.
- Return $\{\mathsf{vk}_i, \mathsf{cert}_i\}_{i\in[n]}, \{\mathtt{msg}_j^\ell, \sigma_j^\ell\}_{j\in[n],\ell\in[m]}$

Figure 4.6: Functionality $T^{\text{HT-FBS}}$ for hardware token behavior in fallback protocol $\Pi^{\text{HT-FBS}}$.

We assume that the party p_j is registered to the global token functionality $T^{\text{HT-FBS}}$ and obtains mpk by querying it with $I = (\text{Get\_key}, \text{sid})$.

Input and next message generation.

- The party p_j on input $(\text{Compute}, \text{sid}, x)$ samples uniform random $R_j \in \{0,1\}^{(2m+4)\kappa}$ and sends $I = (\text{Input}, \text{sid}, x_j, R_j, \Pi^{\text{MPC}})$ to $T_j^{\text{HT-FBS}}$.
- For each $\ell \in \{1, \ldots, m\}$:
  - Upon receiving message X from $T_j^{\text{HT-FBS}}$ check if $X = (\perp, p_{i'})$. If it is then output $(\perp, p_{i'})$ and stop, otherwise send X over broadcast.
  - Collect message $(\mathtt{msg}_i^\ell, \text{sid}, \Pi^{\text{MPC}}, \sigma_i^\ell, \text{vk}_i, \text{cert}_i)$ for round ℓ from each party $p_i \in [n]\backslash\{j\}$ and send $(\text{NextMsg}, \text{sid}, \{\mathtt{msg}_i^\ell, \sigma_i^\ell, \text{vk}_i, \text{cert}_i\}_{i\in[n]})$ to $T_j^{\text{HT-FBS}}$.

Output phase.

- Collect the message $(\mathtt{msg}_i^\ell, \text{sid}, \Pi^{\text{MPC}}, \sigma_i^\ell, \text{vk}_i, \text{cert}_i)$ for round ℓ from each party $p_i \in [n]\backslash\{j\}$ and send $(\text{NextMsg}, \text{sid}, \{\mathtt{msg}_i^\ell, \sigma_i^\ell, \text{vk}_i, \text{cert}_i\}_{i\in[n]})$ to $T_j^{\text{HT-FBS}}$.
- Upon receiving y_j from $T_j^{\text{HT-FBS}}$ output it.

Check-channel. The party p_j inspects all messages that are sent on the channel. If a message $(m, \text{sid}, \Pi^{\text{MPC}}, \sigma, \text{vk}, \text{cert})$ is received from a party p_i check if $\text{Ver}(\text{vk}_i, m||\Pi^{\text{MPC}}||\ell, \sigma) = 1$ and $\text{Ver}(\text{mpk}, \text{vk}||\text{sid}||i, \text{cert}) = 1$ for some $\ell \in [m]$. If it is not, then output $(\perp, p_i)$ and stop.

Figure 4.7: Protocol $\Pi^{\text{HT-FBS}}$ followed by p_j to achieve fallback security.

- If $I = (\text{Input}, \text{sid}, x_1, R_1, \Pi^{\text{MPC}})$ is received from $\mathcal{A}_1$ then do the following.
  - If $\text{ctr}^{\text{sid}} = 0$ then $\text{ctr}^{\text{sid}'} \leftarrow 1$ otherwise output $\perp$ and stop.
  - Set $\mathsf{x} \leftarrow x_1$, $\mathtt{l} \leftarrow 1$ and send $(\mathtt{msg}_1^1, \Pi^{\text{MPC}}, \text{vk}_1, \sigma_1^1, \text{cert}_1)$ to $\mathcal{A}_1$.
- If $I = (\text{NextMsg}, \text{sid}, \{\mathtt{msg}_i^{\ell'}, \sigma_i^{\ell'}, \text{vk}_i', \text{cert}_i'\}_{i\in[n]})$ is received then do the following.
  - If $\ell \neq \mathtt{l}$ or $\mathtt{l} > m$ output $\perp$ and stop, otherwise continue with the following steps.
  - For all $j \in [n]$ if $\mathtt{msg}_j^\ell \neq \mathtt{msg}_j^{\ell'}$ or $\sigma_j^\ell \neq \sigma_j^{\ell'}$ or $\text{cert}_i' \neq \text{cert}_i$ or $\text{vk}_i' \neq \text{vk}_i$ then output $(\perp, p_i)$ and stop.
  - Set $\mathtt{l} \leftarrow \mathtt{l} + 1$.
  - If $\ell \leq m$ then send $(\mathtt{msg}_1^1, \text{sid}, \Pi^{\text{MPC}}, \sigma_1^1)$ to $\mathcal{A}_1$.
  - If $\ell = m + 1$ then send x to $\mathcal{F}$. Upon receiving y_1 from $\mathcal{F}$ send y_1 to $\mathcal{A}_1$.

The proof relies on the observation that (unless the adversary breaks the security of the strong signature scheme), then the adversary just acts as an observer on the channel. That is, the corrupted party $\mathcal{A}_1$ can only inspect the messages generated by $T_1^{\text{HT-FBS}}$ and the messages

received on the channel which are honestly generated using $T^{\text{HT-FBS}}$.[11] For this reason $\mathcal{S}_1, \ldots, \mathcal{S}_n$ can run a simulator S of Π^{MPC} that works when there are no corrupted parties. We recall that the behavior of the corrupted parties cannot influence the output of S as long as the signature scheme is secure.

□

Lemma 10. *Let Π^{MPC} be a protocol that GUC-realizes the n-party functionality $\mathcal{F}^{\text{AB}}$ with* $\text{AB} \in \{\text{IDA}, \text{UNA}\}$ *that exclusively uses $\mathcal{B}$ as a resource. Let $\bar{\mathcal{G}} = \overline{T}^{\text{HT-FBS}}$ and $\mathcal{R} = \mathcal{B}$ then* $\forall \mathcal{A}\ \exists \text{Sim}\ \forall \mathcal{Z}\ \text{EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\text{HT-FBS}},\mathcal{A},\mathcal{Z}} \approx \text{EXEC}^{\bar{\mathcal{G}},\mathcal{F}^{\text{AB}}}_{\text{Sim},\mathcal{Z}}$

Proof. This proof follows immediately from the GUC security of Π^{MPC}. Indeed, we note that the token functionalities simply run Π^{MPC} even in the case that a unsigned message is received (or a message is not received at all). This also enables identifiable abort (unanimous abort) if Π^{MPC} enables it.

□

Having shown the above lemmas, we prove that the fallback protocol $\Pi^{\text{HT-FBS}}$ indeed achieves the property of identifiable abort.

Theorem 6. *Let $\bar{\mathcal{G}} = T^{\text{HT-FBS}}$ be the setup, $\mathcal{R} = \mathcal{B}$ and $\mathcal{F}$ be n-party resources where $\mathcal{F}$ is a CP-well-formed functionality. Then, the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol $\Pi^{\text{HT-FBS}}$ (described by Fig. 4.7) GUC realizes $\mathcal{F}^{\text{IDA}}$.*

Proof. The only way that an adversary has to misbehave is by sending an invalid signature over the channel. This behavior can be detected by any party that has the verification keys of the token functionalities. Moreover, the first party that sends a unsigned message is identified as corrupted. In all the schemes that we have described in this section, the broadcast channel is used exclusively to run the protocol. Hence, once the protocol has been executed the broadcast channel is closed (no messages can be send on that channel). We note that this is not a limitation since the notion of CP tolerates composability. Hence, multiple broadcast channels might be available at the same time.

□

4.2.3.1 Remark on fully-compromised tokens

As discussed in the introduction, to achieve fallback security against fully-compromised (that is, untrusted, non-tamper-resilient, non-isolated) tokens, any computation performed by the token must instead be done through a 2-party protocol between the token and the party holding it. To be specific, this 2-party protocol will implement the following functionality: On request to compute: (1) if round $\ell = 0$: Take input x_j from party p_j and $K_0, \text{mpk}, \text{msk}$ from p_j's token. It stores $x_j, K_0, \text{mpk}, \text{msk}$. (2) If $\ell \leq m$: Take input the messages from the current round from p_j, if a message fails to authenticate then the token stops and p_j aborts. (3) Finally, output the next message $(\mathtt{msg}_j^\ell, \cdots)$ as the token would, to p_j.

[11] If that is not the case then either the protocol would abort, or a reduction to the security of the signature scheme can be done.

4.2.3.2 Remark on malicious broadcast

We also give intuition on what happens if our authenticated broadcast resource is malicious. First, we will invariably lose CP, since a malicious broadcast resource can provide a private communication channel for corrupt parties, trivially allowing collusion. However, like with a malicious mediator in [9], we still achieve (G)UC security without unanimous abort/fairness. This is an interesting feature of our model and results: our security degrades more gradually with respect to the underlying assumptions than with a corrupted mediator. When only the tokens are compromised but authenticated broadcast is honest, then we have (G)UC security with identifiable abort—impossible with a corrupt mediator. If broadcast is malicious then we lose identifiable/unanimous abort and fairness—e.g., the malicious broadcast can pass an unauthenticated invalid message to just one honest party, making only this party abort, and it can also refuse to pass the output to honest parties. Intuitively, security is preserved even with malicious broadcast since messages from the tokens are encrypted and authenticated. If tokens are in addition compromised, then our fallback protocol (when based on a MPC protocol with (G)UC security in presence of a malicious broadcast), also preserves security.

4.3 Collusion-preserving MPC for rational adversaries

In the Preliminaries Ch. 2, Sec. 2.6.2, we detailed the rational protocol design (RPD) framework. In this section, we will extend the RPD framework to study the feasibility of implementing functionalities $\mathcal{F}$ in a collusion-preserving way, against incentive-driven attackers that may even choose to abort the protocol. We first detail how to adapt RPD (with monolithic adversaries) to the case of collusion-preserving functionalities (which we call *RPD-CP*), with local adversaries. We then prove that our protocols Π^{HT} and $\Pi^{\text{HT-FBS}}$ disincentivize collusion in this model, when there is a sufficient penalty to the attacker for aborting. Recall that Π^{HT} and $\Pi^{\text{HT-FBS}}$ are CP against non-aborting adversaries, and can (publicly) identify a corrupt party in case of an abort. Finally, we show how this model can be applied in practice, e.g. using the blockchain. In particular, we can abstract the blockchain by means of an ideal functionality that allows the parties to deposit collateral, which can be reclaimed on the agreement of all parties. We provide a protocol Π_{pen} that uses this functionality to concretely penalize an attacker for aborting Π^{HT} or $\Pi^{\text{HT-FBS}}$.

4.3.1 Motivation for using penalization to disincentivize aborts

We observe that the notions of collusion-freeness (and so collusion-preservation) do not capture all the attacks that send subliminal message using the ability to abort. Indeed and adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ could adapt the following strategy in a game of poker. $\mathcal{A}_1$ aborts any time that he holds an Three of Spades in his hand. Clearly, even if $\mathcal{A}_1$ and $\mathcal{A}_2$ are isolated, when protocol *does not* abort, $\mathcal{A}_2$ knows some information about $\mathcal{A}_1$'s cards that the honest parties do not. This attack becomes more interesting in the case of reactive functionalities, where the parties can get intermediate outputs. As a concrete example of a reactive functionality, we can think to use a CP protocol to shuffle a deck of cards and let the parties play poker. A strategy could be

to make corrupt party $\mathcal{A}_1$ abort in the first hand if he does not have an Ace, and abort in the second hand if he does not have a King, and continues the game otherwise. It should be easy to see that with high probability $\mathcal{A}_1$ will be able to communicate more than one bit of information to $\mathcal{A}_2$. We note that this issue arises only because the CP definition allows the parties to abort, which is in general an unavoidable attack. Thus, in our work we choose to disincentivize aborts.

4.3.2 RPD-CP: Tuning RPD to the CP setting

The Attack Model Following the RPD framework [65] (formally described in Sec. 2.6.2), we capture collusion-preservation against incentive-driven attackers, by considering attacks as part of an *attack game* $\mathcal{G}_{\mathcal{M}}$ between a protocol designer $\mathtt{D}$ and attacker $\mathtt{A}$. Here, $\mathtt{D}$ comes up with a protocol Π, and the attacker $\mathtt{A} \in \mathtt{ITM}$ generates a set of adversaries/adversarial strategies $\mathtt{A}(\Pi) = \mathcal{A} = \{\mathcal{A}_i\}_{i\in\mathcal{I}}$, $\mathcal{I} \subseteq [n]$ to attack it. The attacker's utility $u_{\mathtt{A}}$ is then a function of the choice of protocol Π and adversarial strategies $\mathcal{A}$. The attack model $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}})$ (and $v_{\mathtt{D}}$, if we also consider the designer's utility) encompasses the parameters in this game—$\langle\mathcal{F}\rangle$ is the weaker version of the functionality $\mathcal{F}$ we wish to implement. $\langle\mathcal{F}\rangle$ explicitly allows (1) CP to be broken by sending a colluding message to other adversarial parties and (2) the adversarial parties to abort and being identified by all the other parties that are running the protocol. Note that in contrast to monolithic adversaries and simulators, in CP the ideal adversarial parties do not automatically share their views and must use $\langle\mathcal{F}\rangle$ to collude (see. Fig. 4.8). Lastly, the attacker's utility $u_{\mathtt{A}}$ is defined based on a value function $v_{\mathtt{A}}$, which assigns payoffs to events occurring in the ideal world—more details below.

4.3.2.1 Utility of the attacker $\mathtt{A}$

The utility of the attacker $u_{\mathtt{A}}$ is a function mapping protocols and sets of adversaries, i.e. the strategy profile $(\Pi, \mathcal{A})$, to a real number. In our case, utility depends on whether a set of simulators must collude via a weakness in $\langle\mathcal{F}\rangle$ in order to emulate $\mathcal{A}$ in Π, and whether the simulators trigger an abort. More formally: First, we have a value function $v_{\mathtt{A}}$, defined in the attack model, which maps the views of the simulators and environment in the ideal world to a real value. Then, we define the real payoff of a particular $\mathcal{A}$ attacking the protocol, as the minimum payoff over all simulators that can emulate $\mathcal{A}$. Finally, $u_{\mathtt{A}}(\Pi, \mathcal{A})$ is the real payoff of $\mathcal{A}$, maximized over all possible environments $\mathcal{Z}$.

Ideal payoff of a set of simulators In more detail, we define the real-valued random variable ensemble $\{v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ (or $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ for short) as the random variable ensemble resulting from applying value function $v_{\mathtt{A}}$ to the view of the environment $\mathcal{Z}$ and a set of simulators $\mathcal{S} = \{\mathcal{S}_i\}_{i\in\mathcal{I}}$ in the ideal execution. The ideal expected payoff of a particular set of simulators $\mathcal{S}$ with respect to $\mathcal{Z}$ is defined as the expected value: $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z}) = E(v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}})$.

Real payoff of a set of adversaries Recall that given a setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$, a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive (that is, the protocol only uses $\bar{\mathcal{G}},\mathcal{R}$) protocol Π realizes a CP-functionality $\langle\mathcal{F}\rangle$ if, for

The functionality interacts with a set of parties $\mathcal{P} = \{p_1, \ldots, p_n\}$. It maintains a set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and a set of malicious parties $\mathcal{I} \subseteq \mathcal{P}$

- Upon receiving $(\texttt{COLLUDE}, \text{sid}, m)$ from party $p_i \in \mathcal{I}$, send message $(\texttt{SUB\_MSG}, \text{sid}, p_i, m)$ to p_j, for all $p_j \in \mathcal{P} - \{p_i\}$.
- Upon receiving $(\texttt{ABORT}, \text{sid})$ from a party p_i, send $(\texttt{ABORT}, p_i)$ to p_j, for all $p_j \in \mathcal{P}$-$\{p_i\}$ and stop.
- Upon receiving a message that is consistent with the interface of $\mathcal{F}$ act as $\mathcal{F}$ would do acting as a proxy between $\mathcal{F}$ and the parties in $\mathcal{P}$.

Figure 4.8: $\langle \mathcal{F} \rangle$ weakens $\mathcal{F}$ with commands COLLUDE and ABORT.

all $\mathcal{I} \subseteq [n]$, and independent (rather than monolithic) adversaries $\mathcal{A} = \{\mathcal{A}_i\}_{i \in \mathcal{I}}$, there exists a collection of efficiently computable transformations from ITMs to ITMs $\text{Sim} = \{\text{Sim}_i\}_{i \in \mathcal{I}}$ such that the simulator $\mathcal{S}_i = \text{Sim}_i(\mathcal{A}_i)$ emulates $\mathcal{A}_i$. That is, the environment $\mathcal{Z}$ cannot distinguish between the real world with $\mathcal{A}$ and resource $\mathcal{R}$, and ideal world with $\mathcal{S} = \{\text{Sim}_i(\mathcal{A}_i)\}_{i \in \mathcal{I}}$ and $\langle \mathcal{F} \rangle$. Let $\langle \mathcal{F} \rangle$ be a CP functionality and Π be a protocol. Denote $\mathcal{C}_{\mathcal{A}}$ as the class of simulators $\mathcal{S} = \{\mathcal{S}_i\}_{i \in \mathcal{I}}$ that can emulate the adversarial parties $\mathcal{A} = \{\mathcal{A}_i\}_{i \in \mathcal{I}}$ for $\mathcal{I} \subseteq [n]$. That is, for setup $\bar{\mathcal{G}}$ and resource $\mathcal{R}$,

$$\mathcal{C}_{\mathcal{A}} = \Big\{ \text{Sim}(\mathcal{A}) = \{\text{Sim}_i(\mathcal{A}_i)\}_{i \in \mathcal{I}} \;\Big|\; \forall i \in \mathcal{I}\colon \text{Sim}_i \text{ an efficiently computable mapping from ITM to ITM}, \forall \mathcal{Z}\colon \text{CP-EXEC}^{\bar{\mathcal{G}}, \mathcal{R}}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}}, \langle \mathcal{F} \rangle}_{\Pi, \text{Sim}(\mathcal{A}), \mathcal{Z}} \Big\}.$$

The expected payoff of a set of adversaries and environment $(\mathcal{A}, \mathcal{Z})$ is then defined as $U^{\Pi, \langle \mathcal{F} \rangle}_{\mathtt{A}}(\mathcal{A}, \mathcal{Z}) = \inf_{\mathcal{S} \in \mathcal{C}_{\mathcal{A}}} \{U^{\langle \mathcal{F} \rangle}_{I^{\mathtt{A}}}(\mathcal{S}, \mathcal{Z})\}$. The attacker's utility is then maximized over all environments $\mathcal{Z}$, i.e., $u_{\mathtt{A}}(\Pi, \mathcal{A}) := \hat{U}^{\Pi, \langle \mathcal{F} \rangle}_{\mathtt{A}}(\mathcal{A}) = \sup_{\mathcal{Z} \in \mathtt{ITM}} \{U^{\Pi, \langle \mathcal{F} \rangle}_{\mathtt{A}}(\mathcal{A}, \mathcal{Z})\}$.

4.3.2.2 Utility of the protocol designer D

In [65], the attack game is assumed to be zero-sum, i.e. the designer's utility $u_{\mathtt{D}} = -u_{\mathtt{A}}$. To remove this assumption, we follow the methodology of a more recent work [15] to define $u_{\mathtt{D}}$. In more detail, for each $(\Pi, \mathcal{A})$, we must assign utility for the designer using the *same* simulators and environments as those used for the attacker. Let $\mathcal{S}_{\mathtt{A}}$ denote the class of simulators that were used to obtain the utility of the attacker, and $\mathcal{Z}_{\mathtt{A}}$ denote the class of environments maximizing the utility for simulators in $\mathcal{S}_{\mathtt{A}}$. That is,

$$\mathcal{S}_{\mathtt{A}} = \Big\{ \mathcal{S} \in \mathcal{C}_{\mathcal{A}}\colon \sup_{\mathcal{Z} \in \mathtt{ITM}} \{U^{\langle \mathcal{F} \rangle}_{I^{\mathtt{A}}}(\mathcal{S}, \mathcal{Z})\} = u_{\mathtt{A}}(\Pi, \mathcal{A}) \Big\}$$

and

$$\mathcal{Z}_{\mathtt{A}} = \Big\{ \mathcal{Z} \in \mathtt{ITM}\colon \text{for some } \mathcal{S} \in \mathcal{S}_{\mathtt{A}}\ ,\ U^{\langle \mathcal{F} \rangle}_{I^{\mathtt{A}}}(\mathcal{S}, \mathcal{Z}) = u_{\mathtt{A}}(\Pi, \mathcal{A}) \Big\}.$$

Then, let $v_{\mathtt{D}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ and $U_{I^{\mathtt{D}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})$ be defined similar to the payoffs $v_{\mathtt{A}}^{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ and $U_{I^{\mathtt{A}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})$ respectively. Again following the definitions of [15], the real payoff of the designer is $U_{\mathtt{D}}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A},\mathcal{Z}) = \sup_{\mathcal{S}\in\mathcal{S}_{\mathtt{A}}}\{U_{I^{\mathtt{D}}}^{\langle\mathcal{F}\rangle}(\mathcal{S},\mathcal{Z})\}$. The utility of the designer is then

$$u_{\mathtt{D}}(\Pi,\mathcal{A}) := \hat{U}_{\mathtt{D}}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) = \inf_{\mathcal{Z}\in\mathcal{Z}_{\mathtt{A}}}\{U_{\mathtt{D}}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A},\mathcal{Z})\}.$$

We can extend the attack model with the value function of the designer $v_{\mathtt{D}}$: $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$.

4.3.2.3 Attack-payoff security with collusion-preservation

Similar to the definition of *attack-payoff secure* in [15,65], which we detailed in the Preliminaries Def. 12, we define collusion-preserving attack-payoff (CPAP). Intuitively, a protocol is CPAP with respect to an attack model $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle, v_{\mathtt{A}})$ if it enjoys security and collusion-preservation under this model. That is, no attacker can gain more utility from running our protocol, than running the dummy protocol that uses a functionality $\mathcal{F}$ as a resource.

Definition 26 (CPAP)**.** Let $\mathcal{M} = (\mathcal{F},\langle\mathcal{F}\rangle, v_{\mathtt{A}})$ be an attack model and Π a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol that realizes $\langle\mathcal{F}\rangle$. We say that Π is *CPAP* in $\mathcal{M}$ if $\sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Pi,\mathcal{A}) \overset{\text{negl}}{\leq} \sup_{\mathcal{A}\in\mathtt{ITM}} u_{\mathtt{A}}(\Phi^{\mathcal{F}},\mathcal{A})$ where $\Phi^{\mathcal{F}}$ is the dummy $\{\bar{\mathcal{G}},\mathcal{F}\}$-hybrid protocol which forwards all inputs to and outputs from functionality $\mathcal{F}$.

To complete our framework, we also define ϵ-subgame-perfect equilibrium from [65], and define *collusion-preserving incentive-compatible* (CPIC) similarly to the definition of *incentive compatible (IC)* in [15], which we detail in the Preliminaries Def. 14. Informally, a strategy profile is an ϵ-subgame-perfect equilibrium if no deviation could improve utilities by more than ϵ. Intuitively, a protocol Π is incentive compatible when even the designer is incentivized to stick with it.

Definition 27 (ϵ-subgame-perfect equilibrium [65])**.** Let $\mathcal{G}_{\mathcal{M}}$ be an attack game. A strategy profile $(\Pi,\mathtt{A}(\Pi))$ is an *ϵ-subgame perfect equilibrium* in $\mathcal{G}_{\mathcal{M}}$ if the following conditions hold: (1) for any $\Pi'\in\mathtt{ITM}^n$, $u_{\mathtt{D}}(\Pi',\mathtt{A}(\Pi')) \leq u_{\mathtt{D}}(\Pi,\mathtt{A}(\Pi)) + \epsilon$, and (2) for any $\mathtt{A}'\in\mathtt{ITM}$, $u_{\mathtt{A}}(\Pi,\mathtt{A}'(\Pi)) \leq u_{\mathtt{A}}(\Pi,\mathtt{A}(\Pi)) + \epsilon$.

Definition 28 (CPIC)**.** Let Π be a $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocol and $\vec{\Pi}$ be a set of polynomial-time $\{\bar{\mathcal{G}},\mathcal{R}\}$-exclusive protocols. We say that Π is *$\vec{\Pi}$-CPIC* in the attack model $\mathcal{M}$ iff for some $\mathtt{A}\in\mathtt{ITM}$, $(\Pi,\mathtt{A}(\Pi))$ is a negl(κ)-subgame perfect equilibrium on the restricted attack game where the set of deviations of the designer is $\vec{\Pi}$.

4.3.3 $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ with incentives

In this section we show that the protocols $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ presented in the previous sections are *CPAP* for a natural class of utilities $u_{\mathtt{A}}$. For simplicity we consider the ideal functionality $\mathcal{F}^f$ for SFE parameterized by the function $f : \mathbb{F}^n \to \mathbb{F}$ (this form is without loss of generality—see,

e.g., [102]), which we assume is a CP-well-formed functionality (Def. 25). We refer the reader to the preliminaries, Ch. 2 for a formal definition of $\mathcal{F}^f$. We consider the following events the value function $v_{\mathtt{A}}$ is concerned with. These are events defined on the views of the environment, the (relaxed) CP-functionality $\langle \mathcal{F}^f \rangle$, and the simulators $\mathcal{S} = \{\mathcal{S}_i\}_{i \in \mathcal{I}}$, given adversaries $\mathcal{A} = \{\mathcal{A}_i\}_{i \in \mathcal{I}}$:

- Define the event $E_{\mathtt{collude}}$ as follows: For some $i \in \mathcal{I}$ and message m, the ith simulator $\mathcal{S}_i$ sends the message $(\mathtt{COLLUDE}, \mathsf{sid}, m)$ to $\langle \mathcal{F}^f \rangle$.
- Define the event $E_{\mathtt{abort}}$ as follows: For some $i \in \mathcal{I}$, party p_i aborts and is identified by all the parties as having aborted.

Now, we define the payoffs assigned by $v_{\mathtt{A}}$ to the events above. Denote by $\gamma_{\mathtt{collude}}$ the utility for the attacker obtained by triggering $E_{\mathtt{collude}}$. Denote by $\gamma_{\mathtt{abort}}$ the penalty incurred as result of a malicious party being identified by the honest parties as an aborting party. Then, the utility of the attacker is:

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z}} \Big\{ \inf_{\mathcal{S} \in \mathcal{C}_{\mathcal{A}}} \big\{ \gamma_{\mathtt{collude}} \Pr[E_{\mathtt{collude}}] - \gamma_{\mathtt{abort}} \Pr[E_{\mathtt{abort}}] \big\} \Big\}.$$

Our protocol satisfies CPAP security under the condition that the penalty of being identified as having aborted is greater than the gain from sending a colluding message. In the next section, we will discuss a penalization scheme that makes these penalties concrete.

Theorem 7. *Let $\mathcal{F}^f$ be an ideal CP-well-formed functionality (Def. 25), and $\langle \mathcal{F}^f \rangle$ be as defined in Fig. 4.8. Let $v_{\mathtt{A}}$ be as defined above, for any $\gamma_{\mathtt{collude}}$ and $\gamma_{\mathtt{abort}}$ such that $\gamma_{\mathtt{abort}} > \gamma_{\mathtt{collude}}$. Then the protocol $\Pi^{\mathtt{HT}}$ described in Sec. 4.2 (and the protocol $\Pi^{\mathtt{HT\text{-}FBS}}$ described in Sec. 4.2.3) is* CPAP secure *in the attack model $\mathcal{M} = (\mathcal{F}^f, \langle \mathcal{F}^f \rangle, v_{\mathtt{A}})$.*

Proof sketch. To prove this theorem we rely on the observation that $\Pi^{\mathtt{HT}}$ ($\Pi^{\mathtt{HT\text{-}FBS}}$) is collusion-preserving for $\mathcal{F}^f$ as long as nobody aborts. Moreover, $\Pi^{\mathtt{HT}}$ and $\Pi^{\mathtt{HT\text{-}FBS}}$ achieve (publicly) identifiable abort, since the only way subliminally communicate is by sending a message which is incompatible with the protocol description. Given the way we have set the payoffs, it is always inconvenient for the adversary to trigger the collusion event $E_{\mathtt{collude}}$, as it causes the abort event $E_{\mathtt{abort}}$. □

Full proof. Our main observation is that $\Pi^{\mathtt{HT}}$ ($\Pi^{\mathtt{HT\text{-}FBS}}$) is collusion-preserving for $\mathcal{F}^f$ as long as nobody aborts. That is, the only way to do subliminal communication in $\Pi^{\mathtt{HT}}$ (and in $\Pi^{\mathtt{HT\text{-}FBS}}$) is by sending a message which is incompatible with the protocol description thus causing the protocol to abort. Given this utility function, it is always inconvenient for the adversary to trigger the collusion event $E_{\mathtt{collude}}$, as it causes the abort event $E_{\mathtt{abort}}$. We observe that in the case where we also consider the utility of the designer we can prove that our protocol is CPIC (according to Def. 28) in the case there the utilities of the designer are symmetric to the utility of the adversary.

In the case that all the parties are malicious then we rely on the fact that $\langle \mathcal{F}^f \rangle$ is CP-well-formed functionality and we allow the adversarial parties to communicate freely via $\langle \mathcal{F}^f \rangle$ following the same approach proposed in the proof of Theorem 5.

In the case that at least one party is honest then we need to show collection of efficiently computable transformations $\mathsf{Sim} = \{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n\}$ mapping ITMs to ITMs such that for every set of adversaries $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$, and every PPT environment $\mathcal{Z}$ the following holds:

$$\text{CP-EXEC}^{\bar{\mathcal{G}},\mathcal{R}}_{\Pi^{\text{HT}},\mathcal{A},\mathcal{Z}} \approx \text{CP-EXEC}^{\bar{\mathcal{G}},\langle\mathcal{F}^f\rangle}_{\phi,\mathsf{Sim}(\mathcal{A}),\mathcal{Z}}$$

The simulators are equal to the simulators showed in the proof of Theorem 5 except for the following details. If $\mathcal{S}_i$ receives an invalid message m from $p_i^\star$ (i.e. a message that would yield an honest party to abort) then $\mathcal{S}_i$ sends $(\texttt{COLLUDE}, \mathsf{sid}, m)$ to $\langle\mathcal{F}^f\rangle$ and stops. This situation captures the ability of the corrupted parties to interact with each others at the price of being detected by the honest parties. Indeed, we recall that Π^{HT} enjoys the identifiable abort property. The payoff we have defined ensures that an adversary is never incentivized to abort (i.e. break the CP property). Given that we have proved in Theorem 5 that the if no party aborts then Π^{HT} is collusion-preserving then we can claim that Π^{HT} is CPAP. The same arguments can be applied to $\Pi^{\text{HT-FBS}}$.

□

4.4 Realizing the incentives

The goal of this section is to create a penalization scheme that translates the utilities defined above to concrete (monetary) values. Below, we describe a simple protocol Π_{pen} that realizes a CP functionality $\mathcal{F}$ assuming no aborts, and disincentivizes aborts with penalization. As we show in Theorem 8 below, Π_{pen} achieves CPAP, and is incentive compatible for the designer (i.e. is CPIC), assuming the existence of one honest party. The protocol assumes a functionality $\mathcal{F}_{\mathsf{pen}}$, which allows each party to deposit a collateral of amount d (a parameter of the functionality). Parties can reclaim their collateral *if and only if* all parties send the functionality a `RECLAIM` message ($\mathcal{F}_{\mathsf{pen}}$ is defined formally in Fig. 4.9). In Π_{pen}, honest parties only send `RECLAIM` if they do not detect an abort during execution of Π^{HT} or $\Pi^{\text{HT-FBS}}$. The protocol Π_{pen} works as follows.

1. Each party sends $(\texttt{DEPOSIT})$ to $\mathcal{F}_{\mathsf{pen}}$, to deposit a collateral of amount d. If the functionality returns $(\texttt{DEPOSIT}, 1)$, proceed. Otherwise, stop.

2. Run Π^{HT} (resp. $\Pi^{\text{HT-FBS}}$) (possibly multiple times for reactive functionality, using secret sharing to maintain secret intermediate state) on the broadcast channel $\mathcal{B}$.

3. If Π^{HT} (resp. $\Pi^{\text{HT-FBS}}$) did not abort, then all parties send $(\texttt{RECLAIM})$ to $\mathcal{F}_{\mathsf{pen}}$. $\mathcal{F}_{\mathsf{pen}}$ ensures that parties receive their deposits back if and only if everyone sends $(\texttt{RECLAIM})$.

We define below a class of utilities for the above protocol, making the arguably natural assumption that the attacker (who chooses the adversarial strategies for all corrupted parties) cares about the sum of deposits lost/gained by all corrupt parties, and similarly, the protocol designer cares about the deposits of all honest parties. Let $E_{\texttt{collude}}$ be the event the simulator sends a

The functionality is parameterized by a deposit amount d, and a tuple of initial balances of each party, Balance $= (b_1, \ldots, b_n)$.

- Upon receiving $(\texttt{DEPOSIT})$ from party p_i: If $b_i \geq d$:

  1. In this round, if all $p_j \in \mathcal{P}$ have sent $(\texttt{DEPOSIT})$, and $b_j \geq d$, then continue. Otherwise, send $(\texttt{DEPOSIT}, 0)$ to each party $p_i \in \mathcal{P}$ and stop.
  2. Deposit collateral for each party: for each party $p_i \in \mathcal{P}$: $b_i \leftarrow b_i - d$
  3. Send the message $(\texttt{DEPOSIT}, 1)$ to each party $p_i \in \mathcal{P}$.

- Upon receiving $(\texttt{RECLAIM})$ from party $p_i \in \mathcal{P}$:

  1. In this round, if received $(\texttt{RECLAIM})$ from all $p_i \in \mathcal{P}$, continue to return deposits to all parties. Otherwise, return $(\texttt{RECLAIM}, 0)$ and stop.
  2. For each $p_i \in \mathcal{P}$, return the deposit to p_i: $b_i \leftarrow b_i + d$ and send the message $(\texttt{RECLAIM}, 1)$.

Figure 4.9: $\mathcal{F}_{\mathsf{pen}}$ allows the parties to deposit an amount d, which can be withdrawn only if all the parties registered to the functionality send a command RECLAIM.

COLLUDE message in $\langle \mathcal{F} \rangle$; $E^{\mathtt{A}}_{\mathtt{deposit}}(t)$ (resp. $E^{\mathtt{D}}_{\mathtt{deposit}}(m)$) is the event t corrupt (resp. m honest) parties send $(\texttt{DEPOSIT})$ to $\mathcal{F}_{\mathsf{pen}}$ and $\mathcal{F}_{\mathsf{pen}}$ returns $(\texttt{DEPOSIT}, 1)$; $E^{\mathtt{A}}_{\mathtt{reclaim}}(t)$ (resp. $E^{\mathtt{D}}_{\mathtt{reclaim}}(m)$) is the event where t corrupt (resp. m honest) parties receive the message $(\texttt{RECLAIM}, 1)$. Sets $\mathcal{C}_{\mathcal{A}}, \mathcal{S}_{\mathtt{A}}, \mathcal{Z}_{\mathtt{A}}$ are defined in Section 4.3.2.

$$u_{\mathtt{A}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z}} \Big\{ \inf_{\mathcal{S} \in \mathcal{C}_{\mathcal{A}}} \Big\{ \gamma_{\mathtt{collude}} \Pr[E_{\mathtt{collude}}] - \sum_{t \in [n]} td \cdot \Pr[E^{\mathtt{A}}_{\mathtt{deposit}}(t)] + \sum_{t \in [n]} td \cdot \Pr[E^{\mathtt{A}}_{\mathtt{reclaim}}(t)] \Big\} \Big\}$$

$$u_{\mathtt{D}}(\Pi, \mathcal{A}) = \inf_{\mathcal{Z} \in \mathcal{Z}_{\mathtt{A}}} \Big\{ \sup_{\mathcal{S} \in \mathcal{S}_{\mathtt{A}}} \Big\{ - \gamma_{\mathtt{collude}} \Pr[E_{\mathtt{collude}}] - \sum_{m \in [n]} md \cdot \Pr[E^{\mathtt{D}}_{\mathtt{deposit}}(m)] + \sum_{m \in [n]} md \cdot \Pr[E^{\mathtt{D}}_{\mathtt{reclaim}}(m)] \Big\} \Big\}$$

Informally, we show CPIC and CPAP by proving that the strategy profile $(\Pi_{\mathsf{pen}}, \mathcal{A})$, where the $\mathcal{A}$ is the passive adversarial strategy that just follows Π_{pen} (and does not collude/abort), is an equilibrium solution. We observe that corrupt parties know that if they try to gain more utility by colluding (and thus abort), there is at least one honest party to ensure they lose their collateral, which means they have no incentive to deviate from Π_{pen}. The protocol designer also has no incentive to deviate from Π_{pen} if the adversary is passive, which means this strategy profile is an equilibrium.

Theorem 8. *Let $\mathcal{F}$ be an ideal CP-well-formed functionality, and $\langle\mathcal{F}\rangle$ be as defined in Fig. 4.8. Let $v_{\mathtt{A}}$ and $v_{\mathtt{D}}$ be defined in the utility functions above, let $td > \gamma_{\mathtt{collude}}$ where t is the number of corrupt parties, and assume there is at least one honest party. Then, the $\{\bar{\mathcal{G}}, \mathcal{R}\}$-exclusive protocol Π_{pen}, where $\mathcal{R}$ is broadcast and $\bar{\mathcal{G}} = T^{\mathtt{HT}}, \mathcal{F}_{pen}$ (resp. $\bar{\mathcal{G}} = T^{\mathtt{HT\text{-}FBS}}, \mathcal{F}_{pen}$) :*

- *collusion-preservingly emulates the $\{\bar{\mathcal{G}}, \mathcal{F}\}$-exclusive protocol ϕ assuming the adversary does not abort,*
- *is CPAP secure in the attack model $\mathcal{M}$, and*
- *is $\vec{\Pi}$-CPIC in $\mathcal{M}$.*

Where $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v_{\mathtt{A}}, v_{\mathtt{D}})$ and $\vec{\Pi}$ is the set of all protocols.

Proof sketch. To prove Π_{pen} realizes the CP-functionality $\mathcal{F}$, $\mathcal{S}_i$ (the simulator for party p_i) communicates with $\mathcal{F}_{\mathsf{pen}}$ to determine the status of the deposited collateral. To simulate $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$) it follows the simulator from the proof of Theorem 5 (resp. Lemma 9). To prove CPAP, we show that Π_{pen} realizes $\langle\mathcal{F}\rangle$ by letting $\mathcal{S}_i$ in addition use the COLLUDE command in $\langle\mathcal{F}\rangle$ in case the adversary misbehaves during $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$). Since in $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$) collusion always follows an abort, CPAP of Π_{pen} follows from setting the deposit d so that the cost of abort (losing td collateral) is higher than gains from collusion $\gamma_{\mathtt{collude}}$—that is, the adversary cannot gain more utility via triggering weaknesses in $\langle\mathcal{F}\rangle$. Lastly, to prove CPIC, we show that given a passive adversarial strategy $\mathcal{A}$ that just follows Π_{pen}, the designer has no incentive to deviate from choosing Π_{pen}. We show this by observing that $u_{\mathtt{D}}(\Pi_{\mathsf{pen}}, \mathcal{A}) = 0$, which is the maximum utility of the designer (as parties can only reclaim collateral which they have already deposited). □

Full proof. First, we prove that Π_{pen} realizes the CP-functionality $\mathcal{F}$ as long as there is no abort. To emulate Π_{pen}, the simulator $\mathcal{S}_i$ forwards $(\mathtt{DEPOSIT})$ and $(\mathtt{RECLAIM})$ to $\mathcal{F}_{\mathsf{pen}}$ on behalf of corrupt party p_i and returns whatever $\mathcal{F}_{\mathsf{pen}}$ does to p_i. During $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$), $\mathcal{S}_i$ follows the simulator for p_i in $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$). By Theorem 5 (resp. Lemma 9) which proved that $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$) realize $\mathcal{F}$ when there is no abort, $\mathcal{S}_i$ emulates party p_i in Π_{pen} when there is no abort.

Second, we prove that Π_{pen} is CPAP-secure. To do so, we must show that (1) Π_{pen} realizes $\langle\mathcal{F}\rangle$, and (2) the adversary has no incentive to deviate from Π_{pen} in such a way that forces the simulators to use weaknesses in $\langle\mathcal{F}\rangle$ (collude/abort) in order to simulate. To show (1), we construct a simulator $\mathcal{S}_i$ for each corrupt party p_i. First $\mathcal{S}_i$ sends $(\mathtt{DEPOSIT})$ on behalf of p_i (or not, if p_i chooses not to). Then, $\mathcal{S}_i$ is made aware of whether all parties have sent $(\mathtt{DEPOSIT})$ to $\mathcal{F}_{\mathsf{pen}}$, when $\mathcal{F}_{\mathsf{pen}}$ returns the message $(\mathtt{DEPOSIT}, 1)$ or $(\mathtt{DEPOSIT}, 0)$. If $(\mathtt{DEPOSIT}, 0)$ is returned, the protocol stops. To simulate Step 2 of Π_{pen}, we simply do as the simulator for $\Pi^{\mathtt{HT}}$ (resp. $\Pi^{\mathtt{HT\text{-}FBS}}$) would, except if it sees an invalid message m, then $\mathcal{S}_i$ sends $(\mathtt{COLLUDE}, \mathsf{sid}, m)$ to $\langle\mathcal{F}\rangle$. Finally, $\mathcal{S}_i$ sends $(\mathtt{RECLAIM})$ to $\mathcal{F}_{\mathsf{pen}}$ if p_i does, and forwards $(\mathtt{RECLAIM}, 1)$ or $(\mathtt{RECLAIM}, 0)$ from $\mathcal{F}_{\mathsf{pen}}$.

To prove (2), we show that the adversary has no incentive to deviate from Π_{pen} at all. Now, from our utility function $u_{\mathtt{A}}$, we see that the only way the adversary can possibly gain utility by

deviating is by triggering the event $E_{\texttt{collude}}$ (since in the protocol he can only reclaim collateral he has deposited, collusion is the only possible way to gain positive utility). We observe the only time we may need to use COLLUDE is during simulating Step 2 (running Π^{HT} or $\Pi^{\text{HT-FBS}}$). If collusion occurs during Step 2, at least one honest party will refuse to send (RECLAIM). Thus, the attacker achieves payoff of $-td + \gamma_{\texttt{collude}}$ in this scenario (where t is the number of corrupt parties). This means deviation is not profitable and we obtain CPAP, when we set deposit d such that $td > \gamma_{\texttt{collude}}$.

Finally, to achieve CPIC, we must find an attacker A such that $(\Pi_{\mathsf{pen}}, \mathtt{A})$ is a $\nu(\kappa)$-subgame perfect equilibrium where the designer can arbitrarily choose another protocol in $\vec{\Pi}$. Denote Π_{pen} as Π for ease of reading. Let A be the passive attacker who simply follows Π_{pen}. We now prove the two conditions of subgame perfect equilibrium, which are: (1) for any $\Pi' \in \vec{\Pi}$, $u_{\mathtt{D}}(\Pi', \mathtt{A}(\Pi')) \leq u_{\mathtt{D}}(\Pi, \mathtt{A}(\Pi)) + \epsilon$, and (2) for any $\mathtt{A}' \in \mathtt{ITM}$, $u_{\mathtt{A}}(\Pi, \mathtt{A}'(\Pi)) \leq u_{\mathtt{A}}(\Pi, \mathtt{A}(\Pi)) + \epsilon$. We have already shown (2) in the CPAP proof above. For (1), we observe that the maximum utility possible for the designer is zero, as $\mathcal{F}_{\mathsf{pen}}$ only allows the parties to reclaim what they have deposited. We also observe that except for negligible probably (that the adversary breaks e.g. the signature scheme), for the passive attacker A, the designer achieves zero utility if he chooses $\Pi = \Pi_{\mathsf{pen}}$ (everyone deposits collateral, follows the protocol, and then reclaims), thus $u_{\mathtt{D}}(\Pi, \mathtt{A}(\Pi))$ achieves maximum utility (minus negligible utility loss from the negligible probability of collusion via the adversary breaking e.g. signatures). This implies (1).

□

One could implement deposit/reclaim via, e.g., a smart contract-enabled blockchain. In our next subsection, we will further discuss on penalization and on how to avoid that also honest parties are penalized in the case a misbehavior is detected.

4.4.1 Open question: an alternative penalization functionality

The penalization solution Π_{pen} in Sec. 4.4, while simple and satisfying CPAP and CPIC, requires at least one honest party to penalize the misbehaving party even at the cost of his own collateral being lost. In fact, if a corrupt party aborts during protocol execution, then honest parties can increase their utility by ignoring the abort and still allow everyone to reclaim their collateral. In other words, a strategy profile $(\Pi, \mathcal{A})$, where Π is the same as Π_{pen} *except* it allows parties to always reclaim their collateral, and $\mathcal{A}$ is an adversarial strategy that makes a corrupt party collude and abort, is also an equilibrium solution.

Below, we suggest below an alternative penalization functionality $\mathcal{F}_{\mathsf{penalize}}$ (Fig. 4.10) that only punishes misbehaving parties and thus eliminates this unwanted equilibrium. It is clear to see that if only misbehaving parties are punished, honest parties will have no incentive to deviate from the penalization scheme, as they will always be able to reclaim their deposit. In addition, this solution does not require one party to be honest, as the functionality $\mathcal{F}_{\mathsf{penalize}}$ is responsible for verifying whether the protocol aborted and who is responsible. However, this functionality is more complex and is more difficult to realize than $\mathcal{F}_{\mathsf{pen}}$, as discussed below.

Informally, a protocol Π_{penalize} running $\mathcal{F}_{\mathsf{penalize}}$ disincentivizes misbehavior by allowing parties to run a protocol only if all parties have deposited a collateral. Throughout the protocol, it

keeps track of PROTOCOL messages submitted by the parties. A party can reclaim his collateral if he has behaved correctly throughout protocol execution (according to the PROTOCOL messages). Otherwise, if the protocol aborts, by using the publicly identifiable abort property, the functionality ensures that only identified corrupt parties lose their collateral.

Implementing $\mathcal{F}_{\text{penalize}}$ on the blockchain: some caveats. A natural question is whether $\mathcal{F}_{\text{penalize}}$ can be implemented on the blockchain, for example following the approach of [93]. There, parties make deposits on the blockchain before running an MPC protocol, and can only reclaim their deposits if they submit correct messages to the blockchain at each round. In particular, if the MPC protocol has constant number of rounds, [93] only requires a constant number of ledger rounds.

However, a concerning caveat to implementing $\mathcal{F}_{\text{penalize}}$ in the blockchain is that Π^{HT} and $\Pi^{\text{HT-FBS}}$ now run using the blockchain, a (usually) non-CP protocol, as its communication channel. In particular, this means that *within the protocols* Π^{HT} and $\Pi^{\text{HT-FBS}}$, corrupt parties might trivially collude via its communication resource. We contrast this with the situation of Π_{pen}, our penalization protocol in Sec. 4.4. In Π_{pen}, parties run Π^{HT} and $\Pi^{\text{HT-FBS}}$ via broadcast, and do not require the blockchain while executing these protocols. Even if we assume that the blockchain is accessible to some parties at some point during this execution, we can handle the access as an external communication resource. Collusion-preservation ensures that even given some external communication, the correlation between corrupt parties in $\Pi^{\text{HT}}/\Pi^{\text{HT-FBS}}$ does not increase substantially. For example, a corrupt party cannot simply post some secret key on the blockchain that allows him to arbitrarily subliminally communicate with others. In addition to this caveat, another concern is the difficulty keeping track of protocol messages on the blockchain. In general, one cannot guarantee that transactions sent at some round r will be in the blockchain ledger at round r, as is the case in the analyses of, e.g. [17]. In contrast, in $\mathcal{F}_{\text{pen}}$ (Sec. 4.4) protocol messages of Π^{HT} or $\Pi^{\text{HT-FBS}}$ use the broadcast channel instead of the blockchain, and thus are not subject to idiosyncrasies of the blockchain.

4.4.2 Protocol Π_{penalize} for penalization

We describe below Π_{penalize}, which uses the penalization functionality $\mathcal{F}_{\text{penalize}}$ (Fig. 4.10). The only differences between Π_{penalize} and Π_{pen} are that Π_{penalize} runs the entire protocol using $\mathcal{F}_{\text{penalize}}$ to communicate, and it penalizes only corrupt parties for misbehavior. That is, honest parties never lose their collateral when playing Π_{penalize}.

Protocol Π_{penalize}:

- Each party sends $(\texttt{DEPOSIT})$ to $\mathcal{F}_{\text{penalize}}$, to deposit a collateral of amount d. If the functionality returns $(\texttt{DEPOSIT}, 1)$, proceed. Otherwise, stop.
- Run Π^{HT} or $\Pi^{\text{HT-FBS}}$ on $\mathcal{F}_{\text{penalize}}$, by sending/receiving each protocol message m as $(\texttt{PROTOCOL}, m)$ to/from $\mathcal{F}_{\text{penalize}}$.
- All parties send $(\texttt{RECLAIM})$ to $\mathcal{F}_{\text{penalize}}$.

We can thus prove a theorem about CP, CPAP, and CPIC for Π_{penalize}, analogous to Theorem 8, using the same utility functions $u_{\mathtt{A}}$ and $u_{\mathtt{D}}$ defined in Sec. 4.4.

Theorem 9. *The $\{\bar{\mathcal{G}}\}$-exclusive protocol $\Pi_{penalize}$, where $\bar{\mathcal{G}} = T^{\mathtt{HT}}, \mathcal{F}_{penalize}$ (resp. $\bar{\mathcal{G}} = T^{\mathtt{HT\text{-}FBS}}, \mathcal{F}_{penalize}$) satisfies the same theorem statements as Theorem 8, except assuming $d > \gamma_{\mathtt{collude}}$.*

Proof. The proof is exactly the same as Theorem 8, except instead of running on the broadcast channel the protocol runs entirely on $\mathcal{F}_{\text{penalize}}$ (accessible to the simulator, which makes the simulator's job easier than having to simulate broadcast messages in Π_{pen}). For CPAP, an attacker can now trigger $E_{\mathtt{collude}}$ by only having *one* corrupt party abort and lose his collateral. That is, we can have CPAP only if $d > \gamma_{\mathtt{collude}}$ (i.e., the loss of one party's collateral is more than the gain from colluding). As for the designer $\mathtt{D}$, same as in Π_{pen} the maximum designer utility is still zero, and thus the CPIC proof still holds. □

The functionality is parameterized by a deposit amount d, a tuple of initial balances of each party, Balance $= (b_1, \ldots, b_n)$, and a protocol Π which is either $\Pi^{\mathtt{HT}}$ or $\Pi^{\mathtt{HT\text{-}FBS}}$. It keeps track of PState, initialized as $\perp$, which keeps track of all protocol messages $(\mathtt{PROTOCOL}, \cdots)$.

- Upon receiving $(\mathtt{DEPOSIT})$ from party p_i: If $b_i \geq d$:

  1. In this round, if all $p_j \in \mathcal{P}$ have sent $(\mathtt{DEPOSIT})$, and $b_j \geq d$, then continue. Otherwise, send $(\mathtt{DEPOSIT}, 0)$ to each party $p_i \in \mathcal{P}$ and stop.
  2. Deposit collateral for each party: for each party $p_i \in \mathcal{P}$: $b_i \leftarrow b_i - d$
  3. Send the message $(\mathtt{DEPOSIT}, 1)$ to each party $p_i \in \mathcal{P}$.

- For each round of Π, upon receiving $(\mathtt{PROTOCOL}, m)$ from party p_i:

  1. Verify this message via Π and PState. If Π aborts, stop processing PROTOCOL messages.
  2. Send $(\mathtt{PROTOCOL}, m)$ to all parties and store it in PState.

- Upon receiving $(\mathtt{RECLAIM})$ from party p_i, check PState to see if p_i has misbehaved[a]:

  1. If p_i behaved correctly, return the deposit to p_i: $b_i \leftarrow b_i + d$ and send $(\mathtt{RECLAIM}, 1)$ to p_i.
  2. Otherwise, send $(\mathtt{RECLAIM}, 0)$ to p_i.

[a] This can be checked as Π has publicly identifiable abort

Figure 4.10: $\mathcal{F}_{\text{penalize}}$

4.5 Hardware tokens and broadcast vs. mediators

In this section, we motivate our study of collusion-preservation (CP) using authenticated broadcast channels with hardware tokens, instead of the mediator model used in the previous work.

We show (Theorem 10) that authenticated broadcast (Fig. 2.2) with hardware tokens is a *strictly weaker* set of assumptions than an honest mediator in a star-topology network (also called the *mediated model* [9]). Thus, our feasibility results are not immediately implied by existing results in the mediated model.

Perhaps more interestingly, although our assumptions are weaker, they are also better in terms of the fallback guarantees we can achieve. We will show this by contrasting what happens when tokens are broken, versus when the mediator becomes corrupted. Whereas we can still achieve fallback security *with identifiable and unanimous abort* with corruptible hardware tokens, many such important properties become impossible when the mediator can be corrupted—even with honest majority.

Theorem 10. *Hardware tokens with authenticated broadcast is a strictly weaker set of assumptions than an honest mediator in the mediated model.*

Proof. We prove the theorem by combining the two claims below.

Claim 1. *Hardware tokens with authenticated broadcast does not imply an honest mediator in the mediated model.*

Proof of claim: Let $\mathcal{F}$ be a CP functionality where for all $i = 1, 3, 5, \cdots$, the outputs of parties p_i and p_{i+1} are x_{i+1} and x_i respectively. That is, the pair of parties share their input with only each other. Now consider the case where all but one party are corrupted. Let p_i be the only corrupt party which is supposed to share inputs with the one honest party. Suppose in the real world p_i sends the input of the honest party over broadcast, as soon as p_i receives it as output. This means another corrupt party p_j now also knows the input of the honest party. However, in the ideal world, the simulator for p_j cannot simulate this part of the real-world execution, as $\mathcal{F}$ does not allow him to know the honest party's input. Thus, collusion-preservation is broken. On the other hand, there exists a CP protocol via an honest mediator in the mediated model for this adversary (e.g., the mediator simply ignores p_i's invalid message). Thus, an adversary breaking CP under the tokens/broadcast assumptions, does not break CP in the honest mediator/mediated model. This proves the claim.

Claim 2. *An honest mediator in the mediated model implies hardware tokens with authenticated broadcast.*

Proof of claim: An honest mediator (in a star-topology network) can be used as a trusted party to replace hardware tokens/authenticated broadcast. To emulate each party having his own token, the mediator runs an instance of the hardware token's code for each party. As the mediator is assumed to be honest, it is able to emulate any code correctly, while maintaining each token's secret state. To emulate authenticated broadcast, the honest mediator simply sends each message it receives to every other party with the identity of the sender attached.

□

4.5.1 Impossibility results in the mediated model

In this section, we detail one of our main motivations to move away from the mediated model (used by the previous work on CP [9]), mentioned in the introduction. We show that in this model, desirable properties—robustness, fairness, and identifiable abort—are impossible when the mediator may be corrupt. This implies undesirable scenarios—a game of poker implemented in this model can be aborted at any time (for example, when a corrupt party sees he is losing), without honest players being able to detect the cheater. In contrast, recall that in Lemma 10 and Theorem 6, we showed that even if our assumptions on honest hardware tokens are broken, our construction can still achieve fallback security with identifiable and unanimous abort.

To see why these properties are impossible in the mediated model, we recall that in the mediated model, every party communicates only to a central node called the mediator, in a star-topology network. Intuitively, the impossibilities stem from the mediator being able to cut off communication between itself and any party, at any time. Robustness is simple to show to be impossible, since the mediator can simply stop all communication. To break fairness, the mediator can end communication in the protocol after one party receives his output and before another party receives his. This strategy can be used also to break unanimous abort and allow one honest party to receive his output while others do not. Lastly, identifiability is not possible as a corrupt mediator can "frame" an honest party as having aborted, by simply ignoring messages from this party. Since other parties only communicate through the mediator, they cannot identify whether the party or the mediator has misbehaved.

We state and prove these impossibilities formally in the theorems below.

Theorem 11. *Robustness is not possible when the mediator may be corrupted. This holds even when all other parties are honest.*

Proof. The mediator simply does nothing, and thus nothing can be computed. □

Theorem 12. *Consider a protocol in which each party interacts with the mediator a number of times that is polynomial in the security parameter. Then fairness and unanimous abort are not possible when the mediator may be corrupted, unless the output can be computed without the parties' inputs. This holds even if all other parties are honest.*

Proof. Consider in the mediated model three parties: (honest) parties Alice (A) and Bob (B), and the corrupt mediator (M) who has no inputs and some constant as output (e.g., 1). Since the mediator controls when messages are sent during the protocol, we consider the following protocol format without loss of generality: In odd-numbered interactions, A interacts with M; in even-numbered interactions, B interacts with M.

Suppose M does the following: M guesses an interaction number in which only one of A or B has the output (this is always possible since A and B never receive messages at the same time and in the mediated model, A and B cannot communicate except through M), and aborts at the beginning of this interaction, depriving the other party of the output. Since there is a polynomial number of choices, M succeeds with non-negligible probability.

Formally: Suppose the final interaction in the protocol is between A and M. Assuming that fairness is possible, we show that if this protocol has fairness, then A and B can compute their outputs without sending any messages.

Since the final interaction does not involve B, B must have known the output prior to the final interaction. Suppose the corrupt M chooses to abort in the final interaction (Since the number of rounds is polynomial, he chooses to abort at this round with non-negligible probability). By fairness (since B knows the output without the final message), A must also be able to learn the output without the final interaction. This means both A and B learn the output without the final interaction.

Similarly, the second-to-last interaction (between B and M) does not involve A, so A must have known the output prior to this interaction. By fairness, if M chooses to abort in the second-to-last interaction, B must also be able to learn the output without this interaction. Thus both A and B learn the output without the second-to-last round.

Continue the argument for the number of interactions, and A and B both learn the output without sending any messages. □

The proof does not work when the number of interactions is superpolynomial. This is because the mediator will only guess correctly with negligible probability, which interaction to abort at (we allow negligible failure of the fairness property).

Theorem 13. *Identifiability is not possible in the mediated model when the mediator may be corrupted, regardless of the number of honest parties.*

Proof. We show that it is impossible to distinguish between the case when (1) the mediator is corrupt, and (2) a party A aborts at round r. Suppose M is corrupt and does the following: M follows the protocol until round $r-1$. At rounds $\geq r$, M simply ignores (does not send nor receive from) A, and does whatever it is supposed to do in the protocol if A aborts. Since all parties communicate only via M, the messages they receive in this scenario are exactly the same as if the mediator were honest, but a corrupt party A has followed the protocol before round r and stops sending messages after round r. Thus, they cannot correctly identify whether M or A is corrupt in the case of an abort. □

4.6 Chapter summary and future work

In this chapter, we construct a simple decentralized collusion-free and preserving (CF and CP) protocol using hardware tokens and authenticated broadcast, under assumption of non-aborting adversaries. We motivate our model by showing the already restrictive mediated model solution in previous works suffer from the impossibility of various desirable security properties when the mediator can be corrupt. Then, we augment our simple CF/CP protocol to provide fallback security with identifiable and unanimous abort (impossible in the mediated model) when the hardware tokens are compromised and adversaries abort. Furthermore, we present two penalization schemes and prove in the rational protocol design (RPD) framework (adapted to CP) that they disincentivize rational attackers from aborting. We note that, while our solution is presented in context of collusion-preserving games, the idea of CF/CP has been adapted to *cryptographic reverse firewalls* [108]. We leave as an open question on adapting our protocols and penalization schemes to this and other application scenarios.

Chapter 5

Blockchain Consistency Against Rational Attackers

5.1 Introduction

The classical cryptographic analysis of blockchain ledgers establishes worst-case guarantees on their security either by proving central security properties [66, 115], such as *consistency/common-prefix*—the stable parts of the chains held by honest parties are prefixes of one-another—*liveness*—new blocks with recent transactions keep being added–or by proving that the protocol realizes an ideal ledger functionality [17]. Typically such analyses rely on an assumed limitation on the adversary's influence/presence in the system. In particular, the majority of an underlying resource—e.g., hashing power for proof-of-work (PoW)-based protocols such as Bitcoin [111] and Ethereum [34] (before version 2.0), or stake in proof-of-stake (PoS)-based protocols such as Algorand, Ouroboros, and Snow White [16, 45, 51, 92]—is owned/contributed by parties who honestly run the protocol.

Although such an analysis is instrumental for understanding the properties and limitations of the analyzed ledgers and gaining confidence in their security, it does not take into account a fundamental property of such systems, namely that the ledger's state is often associated with some monetary value and therefore the protocol's security might rely on how profitable an attack might be. Thus, in addition to the classical cryptographic analysis of such systems, it is useful to analyze their so-called *economic robustness*, namely their level of protection or susceptibility to attacks by an incentive-driven (also called rational) attacker. Such an analysis can augment the security of these systems, by proving that under a rational assumption, e.g., an incentives model of the attacker, security is maintained even when certain cryptographic assumptions fail. On the other hand, it can indicate that the proven security is fragile by pointing out natural incentives that lead to violating the security assumptions. Additionally, it can offer a higher resolution picture of the systems guarantees—e.g., its tendency to decentralize [32]—and/or more realistic estimates of the parameters associated with its security properties—e.g., relation between the density of

Most of the results in this chapter have been published in [104].

honest blocks (that is, the chain-quality parameter [66]) and the properties of the communication network [62, 112]. Perhaps, even more interesting, it can offer insight on the system's behavior when the main (cryptographic) assumption fails, e.g., when the attacker controls a 51% fraction of the underlying resource of the blockchain protocol.

Motivated by the recent (repeated) 51% double-spending attacks that have drained millions of dollars from popular blockchain-based cryptocurrencies, in this chapter we focus on the game-theoretic analysis of such attacks for Nakamoto-style systems, e.g., Bitcoin, Bitcoin Cash/Gold, Ethereum (Classic), etc. We use the adaptation of the rational protocol design (RPD) framework by Garay *et al.* [65] to blockchains, which was recently proposed by Badertscher *et al.* [15], to analyze the utility of an attacker against these systems as a function of their basic parameters. In the preliminaries Ch. 2, Sec. 2.6.2 we discuss the RPD framework in more detail.

A central question to the relevance for practice of any game-theoretic analysis is to what extent the model and assumed utilities capture the incentives of real world attacks. Indeed, if the utilities are disconnected from reality, they can lead to counter-intuitive statements. We demonstrate an instance of such an artifact in [15] and propose a different class of utilities which is both natural and avoids this artifact. We validate our utility against a range of security parameters matching those of Ethereum Classic, a PoW-based system that fell victim to 51% double-spending attacks. We observe that when the payoff for double-spending is high, attacking is indeed a dominant strategy. That is, predictions of our utility choice match reality. We then use our framework to devise a generic tuning of one of the core parameters of such blockchains—namely, the number cutOff of most-recent blocks needed to be dropped to achieve the so-called common-prefix property with parameter cutOff (cf. [15, 17, 66])—to deter any attacks on consistency by a rational attacker with our utility. Stated differently, we show how an incentive model can serve, possibly in addition to cryptographic assumptions, to find a robust protocol parametrization. This shows the unique property of our model over previous works, that it can be used to improve the economic robustness of such blockchains, and offers a guide to how to "patch" such protocols to avoid future occurrences.

5.1.1 Related works

A number of works have focused on a rational analysis of decentralized ledgers and cryptocurrencies (e.g., [43, 61, 62, 70, 105, 112, 116, 118–120, 122] to mention some). Typically, these works abstract away the computational aspects of cryptographic tools (signatures, hash-functions, etc.) and provide a game which captures certain aspects of the execution that are relevant for the rational analysis. In contrast, RPD uses a cryptographic simulation-based framework to incorporate these computational considerations into the analyzed game, ensuring that predictions about attacker behavior hold for the actual protocol and not only for an idealized version (unless the idealization is obtained via a cryptographic composition argument such as UC). Incorporating such computational considerations within a rational treatment is highly non-trivial (see [46, 65] for a discussion).

The term *51% (double-spending) attack* is defined in [4] as an attack where the adversary gains any majority (not necessarily just 51%) of mining power and reverses transactions in order to double-spend its coins, often by creating a deep fork in the chain. The site CoinDesk keeps

track of news of 51% attacks [1], of which there are quite many: most recently, Verge suffered an attack with 200 days worth of transactions erased in Feb. 2021, and Firo had in Jan. 2021 been in a "hash war" with a 51% attacker. Also recently, Ethereum Classic suffered three 51% attacks in the same month of August, 2020, prompting a solution called MESS to mitigate such attacks which still may not provide robust security [2]. Other recent victims of such attacks include well-known coins such as Bitcoin Gold (Jan 2020), and Bitcoin Cash (May, 2019). A major avenue of 51% double-spending attacks is the use of rented hash power [3]. The site `https://www.crypto51.app/` gives rough estimates on the vulnerability of different coins, based on whether 51% of hashing power can be rented via a service called Nicehash. In some cases, e.g. Bitcoin Gold, it is estimated (at time of writing, Feb. 2021) to only cost a few hundred dollars to have 51% of hashing power for 1 hour.

Previous works have considered the ability of blockchain protocols to recover from 51% attacks. In [13], conditioned on honest majority being satisfied on expectation, Bitcoin was proven to be resilient against a (temporary) dishonest majority. In [14], no such condition is assumed and the authors give concrete recovery bounds as a function of the actual power of the adversary (captured as a budget to go over majority hashing power). We use the latter work for our analysis of the blockchain's security against incentive-driven attackers.

The profitability of 51% double-spending attacks have also been analyzed in previous works. The work of [33] explores these attacks through an economics perspective, and leaving the cost of the attack as a parameter that is computed via simulations. The work of [88] computes probability of attack by modeling attacks as random walk of two independent Poisson counting processes (PCPs). In comparison, our rational analyses are done in the rational protocol design (RPD) framework, where a fork is formally defined as a command in a UC ledger functionality. Another technique proposed is the Markov Decision Process (MDP) model, which is used by both [70] and [82]. In this model, the adversary takes a series of actions relevant to double-spending: adopting or overriding the honest party's chain, waiting, or stopping. Solving the MDP allows these works to reason about the optimal double-spending adversary. While we do not analyze an optimal double-spending adversary, our model is more general. We do not restrict the actions of the adversary, which allows us to analyze conditions under which the protocol is secure against attacks on consistency by *any* incentive-driven adversary. Moreover, since standard MDP solvers cannot solve infinite state MDPs, the MDP is restricted to only consider situations where the chain length is less than some length c [70].

5.1.2 Overview of our contributions

We start by devising a utility in RPD which naturally captures the incentives of an attacker to provoke a double-spending attack. To this direction, we observe that the utility considered in [15] does not capture such an incentive. Intuitively, the reason is that the utility in [15] essentially only considers incentives related to the consensus layer of the protocol. This means that an attacker is rewarded when successfully mining a block, but is not rewarded depending on the block contents—i.e. what kinds of transactions are in the block. Their extension to a utility function to include transaction fees does not apply to double-spending attacks. In this case, the (only) reason to attack the blockchain stems from the existence of a super-polynomial transaction

fee, and assuming a moderate range of fees, they show that no incentive to attack is present. We discuss why super-polynomial quantities are generally problematic in Section 5.3.

Using analyses from [15], it appears that an attacker with these reasonable utility functions (and assuming moderate transaction fees) has no incentive to fork the blockchain. Yet, looking at real-life double-spending attacks, this is clearly not the case. To understand this disparity between the theoretical analysis and reality, we first notice that [15] does not include a payoff that captures the profit from forking (and double-spending) the blockchain. This can be easily rectified by introducing a special payoff that the attacker receives when successfully creating a deep-enough fork (i.e., orphans a sufficiently long valid chain). Intuitively, this payoff corresponds to the utility that the attacker receives when it double-spends transactions by replacing the original longest chain with his own chain.

However, perhaps counter-intuitively, when analyzing Bitcoin [1] with this extended utility function, the attacker is still indifferent between forking and honest mining. We demonstrate this artifact and pinpoint the reason for it: Intuitively, the utility function from [15] (with or without the extra payoff for forking) rewards the attacker by the same amount in all rounds in which it creates (mines) a block. This means that given any adversary that provokes a fork, there is always an honest-mining adversary who achieves more utility without forking by simply accumulating block rewards over a longer period of time. We distill the source of this issue in a property which we call *unbounded incentives*, and demonstrate that any utility which satisfies this property will make any deviation from passive mining a weakly *dominated* strategy.

To avoid the above counter-intuitive artifact and more closely model reality, we then devise a revision of the utility class of [15]. This utility, which satisfies a property we term *limited horizons*—a strong negation of unbounded incentives—has the property that the (actual) rewards of an adversary mining a block diminish with time. This is a natural way to avoid reasoning about extremely "long-lived" adversaries, i.e., that take decisions based on payoffs too far in the future, and captures features which are well-known in utility theory [26]—intuitively, earning \$10 today is more attractive than \$1 million in 100 years, an example of the "St. Petersburg Paradox". We next turn in analyzing the profitability of 51% double-spending attacks, by showing how our revised utility can actually capture them. We provide a range of payoffs for double-spending which would incentivize an attack. Then we visualize our result using concrete parameters estimated from those of Ethereum Classic, for which performing the attack is indeed a dominant strategy. This demonstrates that the above result can explain, in a game-theoretic framework, how recent victims of 51% attacks are vulnerable.

Finally, we discuss whether and how the blockchain protocol can be tuned in order to deter 51% double-spending attacks. In fact, we provide a much stronger tuning, which deters attacks on consistency by any incentive-driven adversary. The tuning depends on the costs (e.g. electricity or cost to rent hashing power), positive payoffs (e.g. block rewards and payoff for causing a fork, from double-spending or otherwise), and protocol parameters (e.g. the difficulty of creating a block). Intuitively, for any combination of these parameters, we show how the window size of the underlying blockchain protocol can be adjusted so that it is not rational for the attacker to

[1] Our analysis uses Bitcoin as a representative example of Nakamoto-style blockchain ledgers, but similarly any blockchain protocol which realizes the ledger from [16, 17] could be analyzed.

deviate from honest behavior. At the core of this results is a lemma that relates the incentive model to an attack pattern, which coupled with the self-healing properties of Nakamoto-style PoW, leads to the desired estimate of a safe parameter. We view this as demonstration of the power of game theory (and in particular the RPD framework) to aid us in fortifying blockchains even when assumptions made by the cryptographic analyses fail.

Chapter organization. Sec. 5.2: Description of artifact of *unbounded incentives* in the previous work's utility function. Sec. 5.3: Modeling attacker incentives with *limited horizons* utility functions. Sec. 5.4: Analyzing 51% double-spending attacks. Sec. 5.5: Mitigating 51% attacks. Sec. 5.6: Concrete numbers used in figures and examples.

5.2 Artifacts of unbounded incentives

In this section, we discuss an artifact of the utility function Equation 2.2 from [15], which we detailed in the preliminaries Ch. 2, Sec. 2.6.2. In the next section, we will eliminate this artifact by altering this utility function. Concretely, we prove that Eq. 2.2 is inappropriate to capture the most realistic situation of attackers that attack the system, e.g., attempt a fork to profit from double-spending. To do so, we prove Lemma 11 and 12, which show this surprising fact: *if running the protocol (semi-)honestly is profitable in expectation, then there is no incentive for an adversary to fork*. This can be proven by using a similar technique as in [15]: Any fixed payoff for forking incurred by the adversary can be offset by an adversary who runs slightly longer (and still polynomially long) but does not fork. This, however, is an artifact of the asymptotic definition and does not reflect real-world incentive-driven attack scenarios, where mining is anticipated to be profitable—otherwise no one would mine—but attackers still perform forking attacks (in particular, in order to double-spend coins). We distill a property of the utility from [15] that is the reason this artifact, which we call *unbounded incentives*, and prove that any utility satisfying this property will suffer from the same artifact. Looking ahead to the following section, we will propose a natural adaptation of this utility function that does not suffer from the above artifact (and where in particular the duration of an attack actually starts to matter).

5.2.1 Demonstrating the artifact

Let us first consider the straightforward adaptation of the utility from Equation 2.2 to model the payoff (e.g. double-spending) an adversary gains by forking the ledger. Define the event K as: There is a round r where the simulator uses the FORK command of the weak ledger functionality $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ (Fig. A.5) that allows the simulator to invoke a fork. Let $\texttt{fpayoff}$ be the payoff for invoking the fork (in order to simulate a real-world adversary). Then, the utility $u_{\mathtt{f}}$ becomes:

$$u_{\mathtt{f}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward} \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right.\right.$$
$$\left.\left. - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] + \mathtt{fpayoff} \cdot \Pr[K] \right\} \right\}. \tag{5.1}$$

Below, we show that for the the utility function $u_{\mathtt{f}}$ above, the Bitcoin protocol $\Pi^{\dot{\mathtt{B}}}$ is *strongly attack-payoff secure* (Def. 16) as long as mining is profitable. Our proof uses a similar proof strategy as [15]. Informally, first we show that the payoff of any polynomial-run-time adversary $\mathcal{A}$ is bounded by a polynomial $p(\kappa)$ of the security parameter; then, we show that there is a passive, front-running adversary whose run-time is also polynomial (albeit bigger than that of $\mathcal{A}$), and who achieves at least $p(\kappa)$ utility.[2]

Lemma 11 (Attack payoff security with forking)**.** *Let $T_{\mathrm{ub}} > 0$ be the upper bound on total number of mining queries per round, $p \in (0, 1)$ be the probability of success of each mining query, and* cutOff $= \omega(\log(\kappa))$ *be the consistency parameter. Let $\mathcal{M}$ be a model whose induced utility u_f has parameters $\mathtt{fpayoff}, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost} \geq 0$. The Bitcoin protocol $\Pi^{\dot{\mathtt{B}}}$ is strongly attack-payoff secure in $\mathcal{M}$ if $p \cdot \mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost} > 0$.*

Proof. Similar to [15] we analyze the modular Bitcoin protocol with access to the state exchange functionality $\mathcal{F}_{\mathrm{StX}}$ (Fig. 2.6), which abstracts the proof-of-work puzzle of Bitcoin. Instead of parties querying the hash function, they mine by querying $\mathcal{F}_{\mathrm{StX}}$, which tells them whether the mining attempt was successful.

Consider a real world execution with a front-running, passive adversarial strategy $\mathcal{A}_1$, who makes q^* queries to $\mathcal{F}_{\mathrm{StX}}$ in total (note that the code of this adversary is in fact static, and the number of queries increases as it is executed for more rounds by the environment), and who does not attempt to fork the chain. Let X_i be the random variable where $X_i = 1$ if and only if his ith query successfully mines a block. Then, the real world payoff of $\mathcal{A}_1$ is

$$R_{\mathcal{A}_1} = \left(\sum_{i=1}^{q^*} X_i \right) \mathtt{breward} \cdot \mathtt{CR} - q^* \cdot \mathtt{mcost}$$

Now, consider any adversary $\mathcal{A}_2$ in the real world. Let Q denote the number of queries made by $\mathcal{A}_2$ in an execution of the protocol under the environment $\mathcal{Z}$, and let P_Q be the associated distribution. Then we define $q := \max Supp(P_Q)$ (where $Supp$ is the support). The expected real world payoff of $\mathcal{A}_2$ is

$$E(R_{\mathcal{A}_2}) \leq q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}$$

[2]We note that for the simple utility function presented in [15] other proof techniques could conclude attack-payoff security without the runtime-extension argument. The main point here is to demonstrate the importance of considering the attack duration in the utility function.

We want to show that $\mathcal{A}_1$ gains more payoff than $\mathcal{A}_2$. We do this by showing that for an appropriate choice of q^*, the payoff of $\mathcal{A}_1$ exceeds the expected payoff of $\mathcal{A}_2$ with overwhelming probability.

Now, let $X = \sum_{i=1}^{q^*} X_i$, then $E(X) = q^*p$. Let δ be such that $(1-\delta)p\cdot\mathtt{breward}\cdot\mathtt{CR}-\mathtt{mcost} > 0$ (this exists by assumption of $p \cdot \mathtt{breward} \cdot \mathtt{CR} - \mathtt{mcost} > 0$ and that $p, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost}$ are constants). We have

$$\Pr(R_{\mathcal{A}_1} < E(R_{\mathcal{A}_2})) = \Pr\left(X < \frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}}\right)$$

We can use Chernoff bound to upper-bound this probability, if $\frac{q^*\mathtt{mcost}+q\cdot\mathtt{breward}\cdot\mathtt{CR}+\mathtt{fpayoff}}{\mathtt{breward}\cdot\mathtt{CR}} < (1-\delta)E(X) = (1-\delta)q^*p$. This inequality is satisfied if we set $q^* = \frac{(q\cdot\mathtt{breward}\cdot\mathtt{CR}+\mathtt{fpayoff})\kappa}{(1-\delta)\mathtt{breward}\cdot\mathtt{CR}-\mathtt{mcost}}$, which is still a polynomial in κ. Thus, by Chernoff bound,

$$\Pr\left(X < \frac{q^*\mathtt{mcost} + q \cdot \mathtt{breward} \cdot \mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward} \cdot \mathtt{CR}}\right) \leq \exp\left(\frac{-\delta^2 q^* p}{2}\right) = \mathsf{negl}(\kappa)$$

To show that analysis of the real world utility is sufficient, we have to prove that this utility is the payoff in the ideal world, minimized over simulators that can simulate the adversary ($\mathcal{A}_1$ or $\mathcal{A}_2$), and maximized over all environments. This is true following the same argument as [15]: A successful simulator must answer the same number of queries as the adversary. Moreover, the number of mining successes must be the same (up to a negligible difference). Otherwise, the environment can distinguish the real and ideal world. □

5.2.2 A first attempt to eliminate the artifact

Although we proved that Bitcoin is strongly attack payoff secure even with a payoff for forking, this is actually not a good sign, as this result does not reflect reality. In reality, attackers do fork blockchains to gain profit via e.g. double-spending transactions. Thus, the fact that we can prove Lemma 11 means that there must be a problem with our assumptions.

Why were we able to prove Lemma 11? It turns out the utility function we used has the weakness that it considers an attacker who does not care about the ephemeral payoff for forking—he can simply obtain more utility via block rewards if he just put in a bit more hashing power for mining. Thus, somewhat counter-intuitively, to model incentives for forking attacks, we must consider utilities that limit the amount of mining an attacker can do.

A first natural instinct may be to incorporate in the utility the (often substantial) initial investment (e.g. cost of buying mining rigs) an attacker must make before being able to participate in the blockchain protocol. This turns out to be not only a natural extension, but also a very simple one. Concretely, we capture this investment as *cost of party corruption*: in order to use party for mining, the adversary needs to corrupt him, which corresponds to acquiring its mining equipment. Formally, for each $g \in \mathbb{N}$ define $C_g^{\mathtt{A}}$ as follows: The maximum number of corrupted parties at any round is g. Let $\mathtt{ccost}(g)$ be the cost of event $C_g^{\mathtt{A}}$, i.e. corrupting g parties—this can be seen as the total cost of mining rigs an attacker obtains. Then we define the utility function:

$$u_{\mathtt{f},\mathtt{c}}(\Pi,\mathcal{A}) := \sup_{\mathcal{Z}\in\mathtt{ITM}}\left\{\inf_{\mathrm{S}^{\mathcal{A}}\in\mathcal{C}_{\mathcal{A}}}\left\{\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}\cdot\mathtt{CR}\cdot\Pr[I^{\mathtt{A}}_{b,r}]\right.\right.$$
$$-\sum_{(q,r)\in\mathbb{N}^2} q\cdot\mathtt{mcost}\cdot\Pr[W^{\mathtt{A}}_{q,r}]$$
$$+\,\mathtt{fpayoff}\cdot\Pr[K]$$
$$\left.\left.-\sum_{g\in\mathbb{N}}\mathtt{ccost}(g)\cdot\Pr[C^{\mathtt{A}}_{g}]\right\}\right\}. \tag{5.2}$$

Interestingly, as we see below, this natural extension is still insufficient to align the model with the reality that forking attacks occur. Indeed, even with this additional cost, we can still prove a result similar Lemma 11. Concretely, the following lemma shows that for $u_{\mathtt{f},\mathtt{c}}$ above, we can prove the statement as the one in Lemma 11 about $\Pi^{\text{₿}}$ being attack-payoff secure by again exploiting the artifact of unbounded incentives.

Lemma 12 (Attack payoff security with forking, with cost of corruption)**.** *Let $T_{\mathrm{ub}} > 0$ be the upper bound on total number of mining queries per round, $p \in (0,1)$ be the probability of success of each mining query, and* cutOff $= \omega(\log(\kappa))$ *be the consistency parameter. Let $\mathcal{M}$ be the model whose induced utility $u_{f,\mathtt{c}}$ has parameters $\mathtt{fpayoff}, \mathtt{breward}, \mathtt{CR}, \mathtt{mcost} \geq 0$, $\mathtt{ccost}(\cdot) : \mathbb{N} \to \mathbb{R}^+$. The Bitcoin protocol is strongly attack-payoff secure in $\mathcal{M}$ if $p\cdot\mathtt{breward}\cdot\mathtt{CR} - \mathtt{mcost} > 0$.*

Proof. The proof goes the same way as in Lemma 11 except here we consider the front-running, passive adversary $\mathcal{A}_1$ who corrupts exactly one party. Let X_i be the random variable where $X_i = 1$ means the ith query yields a succesfully-mined block. Then, the payoff of $\mathcal{A}_1$ is

$$R_{\mathcal{A}_1} = \left(\sum_{i=1}^{q^*} X_i\right)\mathtt{breward}\cdot\mathtt{CR} - q^*\cdot\mathtt{mcost} - \mathtt{ccost}(1)$$

We want to show that the passive strategy gains more payoff than any adversarial strategy $\mathcal{A}_2$. For any adversary $\mathcal{A}_2$: If he corrupts no one, then he has exactly the same payoff as an front-running, passive adversary who corrupts no one so the lemma trivially holds. Thus, let us consider the case if the adversary $\mathcal{A}_2$ corrupts at least one party. Let Q denote the number of queries made by $\mathcal{A}_2$ in an execution of the protocol under the environment $\mathcal{Z}$, and let P_Q be the associated distribution. Then we define $q := \max Supp(P_Q)$. Then, the expected payoff of $\mathcal{A}_2$ is

$$E(R_{\mathcal{A}_2}) \leq q\cdot\mathtt{breward}\cdot\mathtt{CR} + \mathtt{fpayoff} - \mathtt{ccost}(1)$$

Then, the probability

$$\Pr(R_{\mathcal{A}_1} < E(R_{\mathcal{A}_2})) = \Pr\left(X < \frac{q^*\mathtt{mcost} + q\cdot\mathtt{breward}\cdot\mathtt{CR} + \mathtt{fpayoff}}{\mathtt{breward}\cdot\mathtt{CR}}\right)$$

which is the same negligible probability as in the proof of Lemma 11 and the proof follows.

□

5.2.3 The source of the artifact: unbounded incentives

Distilling the issue in above lemmas, we observe that that as long as the adversary keeps accumulating rewards as rounds are added to the protocol—i.e., mining remains profitable—he does not care about the payoff for forking: there always exists a polynomial-time, passively mining strategy that simply gains the same amount of utility by mining a bit more. However, not only do real-life attackers in fact profit from forks, even the assumption on the profitability of mining forever is unrealistic: any attacker is at least limited in time by e.g. the anticipated age of the universe, and cannot, in practice, keep accumulating utility in perpetuity.

Thus, to make accurate prediction about the attackability of a blockchain protocol the utility function must exclude the eternal profitability of passive mining. We generalize this intuition, by defining the notion of *unbounded incentives*: a utility function has *unbounded incentives* if there is an adversarial strategy $\mathcal{A} \in \mathbb{A}_{\mathtt{fr}}$ such that for any polynomial $h(\kappa)$, $\mathcal{A}$ can gain better payoff than $h(\kappa)$. (Conversely, we will say that a utility has bounded incentives if there there is no such passive adversary.).

It is straightforward to verify that the utilities we have seen so far have unbounded incentives, which explains the effect of the artifact exploited in the above lemmas. In fact, in the following there is a simple argument for a generic statement about the strong attack-payoff security of utility functions that have unbounded incentives.

Lemma 13. *Let $\mathcal{M}$ be a model inducing a utility function $u_{\mathtt{A}}$. Assume for any adversary $\mathcal{A}$, in any real execution of the protocol his payoff is polynomially-bounded.* [3] *If $u_{\mathtt{A}}$ has unbounded incentives for a protocol Π, then Π is strongly attack-payoff secure for $\mathcal{M}$.*

Proof. Suppose $u_{\mathtt{A}}$ has unbounded incentives. Then let $\mathcal{A} \in \mathbb{A}_{\mathtt{fr}}$ be the adversary in the definition of unbounded incentives. Then $\mathcal{A}$ is a witness for strong attack-payoff security: For every real execution of another adversary $\mathcal{A}'$, the passive adversary $\mathcal{A}$ can gain better payoff than $\mathcal{A}'$.

□

5.3 A RPD analysis of forks

In this section, we will tune our utility function to avoid the issue of *unbounded incentives* isolated in the previous section. A straw man approach would be to make `fpayoff` a super-polynomial function of the security parameter. But this would imply a very unnatural assumption, which, intuitively, corresponds to ensuring that the polynomially-bounded adversaries are *always* incentivized to fork. This would have the opposite effect and introduce a different artifact: it would make attack-payoff security impossible, and making a 51% attack always a dominant strategy no matter the systems parameters, contradicting the observable fact that many blockchains have not fallen to 51% attacks.

Instead, we make `breward` a function of time. This captures, for example, inflation, or simply that the adversary only plans to stay in the system for a limited amount of time. We refer to this

[3] This is true for the utility function $u_{\mathtt{A}}^{\mathtt{B}}$ in Equation 2.2 (as well as the utility functions we will consider)—no adversary can get payoff that is superpolynomial in the run time of the execution.

adaptation of $u_{\mathtt{f,c}}$ as $u_{\mathtt{buy}}$, which represents the utility function where the corruption cost $\mathtt{ccost}(\cdot)$ models the attacker buying mining rigs. Here, we do not consider the time when modeling the corruption cost, as the cost of a mining rig is relatively stable. We also do not consider time for `fpayoff`, since the "payoff" for a successful fork (i.e., the money stolen by an attacker) can be considered as the fiat currency (e.g. USD) gained from the attacker trading in his coins, before his attack to create a consistency failure on the chain and retrieve these traded coins.

$$u_{\mathtt{buy}}(\Pi, \mathcal{A}) := \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward}(r) \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right. \right. \\ - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \\ + \mathtt{fpayoff} \cdot \Pr[K] \\ \left. \left. - \sum_{g \in \mathbb{N}} \mathtt{ccost}(g) \cdot \Pr[C^{\mathtt{A}}_{g}] \right\} \right\}. \tag{5.3}$$

We also define a version of this utility $u_{\mathtt{rent}}$ (Eqn. 5.4), which models the attacker renting hashing queries by replacing `mcost` with parameter `rcost` (rent cost) and setting $\mathtt{ccost}(\cdot) = 0$. Renting especially has been observed in real attacks, such as the August 2020 attacks on Ethereum Classic [3].

$$u_{\mathtt{rent}}(\Pi, \mathcal{A}) = \sup_{\mathcal{Z} \in \mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r) \in \mathbb{N}^2} b \cdot \mathtt{breward}(r) \cdot \mathtt{CR} \cdot \Pr[I^{\mathtt{A}}_{b,r}] \right. \right. \\ - \sum_{(q,r) \in \mathbb{N}^2} q \cdot \mathtt{rcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}] \\ \left. \left. + \mathtt{fpayoff} \cdot \Pr[K] \right\} \right\}. \tag{5.4}$$

Note that while `breward` is a function of time, we let the cost of a mining query, that is `mcost`/`rcost`, remain constant. We do so to model the attacker's anticipated monetary budget to launch and maintain an attack, such as the costs for renting a certain amount of hashing power (which are generally paid up-front), or cost of electricity (which realistically appears to be relatively stable). Further, the parameter `fpayoff` should be seen as an abstract excess payoff for the attacker arising from forking that is able to capture various use-cases. In the prototypical (double-spend) example where the attacker sells some coins for fiat currency and later tries to regain the coins with a successful attack, it corresponds to this extra fiat inflow gained prior to attacking the blockchain. We note that the utility functions could be tweaked to allow for all parameters to be time-dependent without changing the results qualitatively as long as the relations among the parameters required by the definitions and theorems (which are time-dependent in our treatment already) still hold.

To capture realistic utilities, we restrict to instances of our utility function which satisfy what we call *limited horizons* (Definition 29). Roughly, limited horizons constrains utilities by requiring that passive mining eventually becomes unprofitable. Recall that in light of the St. Petersburg Paradox discussed in the introduction, rational parties become increasingly reluctant to invest some monetary budget for potential rewards gained only later in a randomized process (e.g. due to uncertainty about the future or other specific utility-relevant considerations like relative inflation between several quantities). We cast this general idea as a rather simple condition based on our utility function.

After defining limited horizons, in Section 5.3.1, we will first address a technical challenge imposed when payoff-parameters in the utility functions are non-constant. Then, in Section 5.3.2 we show that limited horizons implies bounded incentives (i.e., the opposite of unbounded incentives) through Lemma 16. More precisely, limited horizon is a strong negation[4] of unbounded incentives. Looking ahead, we will prove that when utilities have limited horizons, there is always a large enough payoff for forking such that (strong) attack-payoff security is broken. Informally, a utility function $u_{\texttt{buy}}$ (resp. $u_{\texttt{rent}}$) has limited horizons if there is a time limit after which passive mining becomes unprofitable.

Definition 29 (Limited horizons)**.** We say $u_{\texttt{buy}}$ in Equation 5.3 (resp. $u_{\texttt{rent}}$, formally defined in Equation 5.4), parameterized by $\texttt{breward}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $\texttt{mcost}, \texttt{fpayoff} \geq 0$, and non-decreasing function $\texttt{ccost}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$ (resp. $\texttt{breward}(\cdot) : \mathbb{N} \to \mathbb{R}_{\geq 0}$, $\texttt{rcost}, \texttt{fpayoff} \geq 0$) satisfies *limited horizons* (resp. *limited horizons with renting*) if $\texttt{breward}(\cdot)$ is a non-increasing function such that $\exists x \in \mathbb{N} : \ p \cdot \texttt{CR} \cdot \texttt{breward}(x) < \texttt{mcost}$.

Remark. Technically, $u_{\texttt{rent}}$ is a special case of the case of $u_{\texttt{buy}}$ (since the utilities are the same if we set $\texttt{mcost} = \texttt{rcost}$ and set $\texttt{ccost}(\cdot) = 0$); however semantically they are different: $\texttt{rcost}$ represents the cost of renting a hashing query, which usually is much higher than $\texttt{mcost}$ which represents the cost (e.g. electricity) of an adversary mining with his own equipment. Nevertheless, to reduce redundancies in the technical sections, we will analyze the utility $u_{\texttt{buy}}$ in Equation 5.3 (with a general $\texttt{ccost}(\cdot)$, including when $\texttt{ccost}(\cdot) = 0$), and state the results for the renting case as corollaries.

5.3.1 Addressing technical issue of non-constant payoff for block rewards

In this section, we address a technical issue with considering a non-constant $\texttt{breward}$—recall that in limited horizons, $\texttt{breward}$ is a non-increasing function of time/round number. By our definition (which follows that of [15]), the event $I^{\texttt{A}}_{b,r}$ happens when b blocks are placed into the ledger of some honest party. This is intuitive—the block reward should be given only when the block is "confirmed" to be in the ledger. However, there is a delay between when a block is broadcasted, and when it makes it into the common prefix of an honest chain. This delay is

[4]Note that the *strong negation* of an assertion A is one which implies $\neg A$, but is not necessarily implied by $\neg A$.

a random variable which depends on the amount of (honest and corrupt) hashing power in the protocol, the network delay, and the adversary's strategy. Fortunately, we can lower and upper bound such a delay (which we denote by t_{lb}, t_{ub} respectively), as we show in the following lemma. This will in turn allow us to avoid the complication of analyzing when blocks enter the ledger state and instead analyze when locks broadcasted by the adversary to honest parties (whose events are easier to analyze). Note that we choose to analyze time-of-block-broadcast, instead of time-of-block-creation, since the adversary may choose to withhold successfully-mined blocks instead of broadcasting them immediately, making time-of-broadcast more suitable for incorporating such adversarial strategies.

We first define a useful quantity $t_\delta^\Delta(q)$. As we will see, this quantity, which is derived from the *chain growth* property of Nakamoto-style blockchains, is the maximum time for honest chains to grow by cutOff blocks, given that in each round there are at least q honest mining queries.

Definition 30 (Maximum time to grow cutOff blocks)**.** For network delay Δ, and $p, \delta \in (0,1)$, we denote $t_\delta^\Delta(q) := \frac{\mathsf{cutOff}}{(1-\delta)\gamma}$, where $\gamma := \frac{h}{1+h\Delta}$ and $h := 1-(1-p)^q$.

Let $t_{lb} := 0$ and $t_{ub} := t_\delta^\Delta(\mathsf{T}_{\mathrm{ub}})$. Let $B_{b,r}^{\mathtt{A}}$ denote the event: At round r, the adversary broadcasts b blocks made by parties that are corrupted at the time of the blocks' creation, and which are part of the longest chain at round r. Let u_{buy}^h be u_{buy} except $\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}(r)\cdot\mathtt{CR}\cdot\Pr[I_{b,r}^{\mathtt{A}}]$ (which considers time of block confirmation) is replaced with $\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}(r+t_{lb})\cdot\mathtt{CR}\cdot\Pr[B_{b,r}^{\mathtt{A}}] = \sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}(r)\cdot\mathtt{CR}\cdot\Pr[B_{b,r}^{\mathtt{A}}]$ (which considers time of block broadcast). Similarly, let u_{buy}^l replace the same term in u_{buy} with $\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}(r+t_{ub})\cdot\mathtt{CR}\cdot\Pr[B_{b,r}^{\mathtt{A}}]$.

$$u_{\mathsf{buy}}^h(\Pi^{\ddot{\mathtt{B}}},\mathcal{A}) = \sup_{\mathcal{Z}\in\mathtt{ITM}}\left\{\inf_{\mathrm{S}^{\mathcal{A}}\in\mathcal{C}_{\mathcal{A}}}\left\{\begin{aligned}&\sum_{(b,r)\in\mathbb{N}^2} b\cdot\mathtt{breward}(r+t_{lb})\cdot\mathtt{CR}\cdot\Pr[B_{b,r}^{\mathtt{A}}]\\ &\quad-\sum_{(q,r)\in\mathbb{N}^2} q\cdot\mathtt{mcost}\cdot\Pr[W_{q,r}^{\mathtt{A}}]\\ &\quad+\mathtt{fpayoff}\cdot\Pr[K]\\ &\quad-\sum_{g\in\mathbb{N}}\mathtt{ccost}(g)\cdot\Pr[C_g^{\mathtt{A}}]\end{aligned}\right\}\right\}.$$

and

$$u^l_{\mathtt{buy}}(\Pi^{\dot{\mathtt{B}}}, \mathcal{A}) = \sup_{\mathcal{Z}\in\mathtt{ITM}} \left\{ \inf_{\mathrm{S}^{\mathcal{A}}\in\mathcal{C}_{\mathcal{A}}} \left\{ \sum_{(b,r)\in\mathbb{N}^2} b \cdot \mathtt{breward}(r+t_{ub}) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,r}] \right.\right.$$
$$- \sum_{(q,r)\in\mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,r}]$$
$$+ \mathtt{fpayoff} \cdot \Pr[K]$$
$$\left.\left. - \sum_{g\in\mathbb{N}} \mathtt{ccost}(g) \cdot \Pr[C^{\mathtt{A}}_g] \right\} \right\}.$$

The following lemma tells us that instead of analyzing the utility function defined on when a block is confirmed in the ledger we can instead approximate by only analyzing when a block is broadcasted. This will be helpful in our proof of Lemma 16 on the utility of the optimal front-running, passive adversary.

Lemma 14 (Translating time-of-block-confirmation to time-of-block-broadcast: $u^h_{\mathtt{buy}}$ and $u^l_{\mathtt{buy}}$)**.** *For any utility function satisfying limited horizons (in fact, we only require that $\mathtt{breward}(\cdot)$ is a non-increasing function), satisfies the following: For all adversaries $\mathcal{A}$, and front-running, passive $\mathcal{A}'$,*

$$u_{\mathtt{buy}}(\Pi^{\dot{\mathtt{B}}}, \mathcal{A}) \leq u^h_{\mathtt{buy}}(\Pi^{\dot{\mathtt{B}}}, \mathcal{A}) + \mathsf{negl}(\kappa) \quad \textit{and}$$
$$u_{\mathtt{buy}}(\Pi^{\dot{\mathtt{B}}}, \mathcal{A}') + \mathsf{negl}(\kappa) \geq u^l_{\mathtt{buy}}(\Pi^{\dot{\mathtt{B}}}, \mathcal{A}').$$

Proof. The first inequality is obvious: By limited horizons, giving block rewards using time-of-block-broadcast (i.e., $u^h_{\mathtt{buy}}$) gives the attacker a higher payoff.

The second inequality: Let the environment be one which maintains T_{ub} parties in each round after r. The bound follows then from the chain-growth lower bound which states the minimum chain length increase during a time period, depending on the honest parties' hashing power and the network delay (cf. [18, 115]). We state one version of the chain growth lemma below for completeness.

Lemma 15 (Chain growth, Lemma 7.11 [18])**.** *For any miner p_i, round number $r \geq 0$, $t \geq 1$, success probability of one mining query $p \in (0,1)$, network delay Δ, and $\delta \in (0,1)$, let $\gamma := \frac{h}{1+h\Delta}$ where $h := 1-(1-p)^q$ and q is the minimum total number of honest mining queries in any round during the interval $[r, r+t]$.*

Suppose p_i is honest in round r, and the longest state received or stored by p_i in round r has length ℓ. Then, in round $r+t$, it holds, except with probability at most $\mathsf{negl}(\gamma t)$[5], that the length of the longest state (received or stored) of at least one honest miner p_j in that round has length at least $\ell + T$ if $t \geq \frac{T}{(1-\delta)\cdot\gamma}$.

This concludes the proof. □

[5] In fact, at most $\exp\left(-\frac{\delta^2 t\gamma}{2+\delta}\right)$.

5.3.2 Optimal utility of front-running, passive adversaries

We show in this section if a utility satisfies limited horizons, then it also satisfies bounded incentives. We do so by proving the following optimal utility of a passive, front-running adversary. We define $u^{\mathsf{h}}_{\mathsf{honest}}$ and $u^{\mathsf{l}}_{\mathsf{honest}}$ which, as we will see in Lemma 16 below, are the upper and lower bounds on the optimal utility obtained by a front running, passive adversary in $\Pi^{\hat{\mathsf{B}}}$.

Definition 31 (Bounds $u^{\mathsf{h}}_{\mathsf{honest}}$ and $u^{\mathsf{l}}_{\mathsf{honest}}$ for optimal front-running, passive adversary)**.** We define the quantity

$$u^{\mathsf{h}}_{\mathsf{honest}}(\mathtt{breward}, \mathtt{CR}, \mathtt{mcost}, \mathtt{ccost}) := g \cdot p \cdot \mathtt{CR} \cdot \sum_{x=1}^{t} [\mathtt{breward}(x + t_{lb}) - \mathtt{mcost}] - \mathtt{ccost}(g)$$

with

$$t := \arg\max_{x \in \mathbb{N}} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{lb}) \geq \mathtt{mcost}),$$
$$g := \arg \max_{g \in [0, \mathsf{T}_{\mathrm{ub}}]} (mg - \mathtt{ccost}(g)),$$
$$\text{for } m := \sum_{x=1}^{t} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{lb}) - \mathtt{mcost}),$$

and the quantity

$$u^{\mathsf{l}}_{\mathsf{honest}}(\mathtt{breward}, \mathtt{CR}, \mathtt{mcost}, \mathtt{ccost}) := g \cdot p \cdot \mathtt{CR} \cdot \sum_{x=1}^{t} [\mathtt{breward}(x + t_{ub}) - \mathtt{mcost}] - \mathtt{ccost}(g)$$

with

$$t := \arg\max_{x \in \mathbb{N}} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{ub}) \geq \mathtt{mcost}),$$
$$g := \arg \max_{g \in [0, \mathsf{T}_{\mathrm{ub}}]} (mg - \mathtt{ccost}(g)),$$
$$\text{for } m := \sum_{x=1}^{t} (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x + t_{ub}) - \mathtt{mcost}).$$

We simplify the above upper and lower bounds on the optimal front-running, passive adversaries as $u^{\mathsf{h}}_{\mathsf{honest}}$ and $u^{\mathsf{l}}_{\mathsf{honest}}$, when the parameters to the utility function are clear from context. As discussed before, although we prove the optimal passive adversary for $u_{\mathtt{buy}}$, the renting case for utility $u_{\mathtt{rent}}$ is a direct corollary by setting $\mathtt{ccost}(\cdot) = 0$ and $\mathtt{mcost} = \mathtt{rcost}$.

Intuitively, the following lemma is established by proving that (1) due to limited horizons, there is a fixed time t after which an optimal passive adversary will not mine, and (2) it is optimal for a passive adversary to corrupt parties statically. Then, we can re-write the utility of a front-running, passive adversary as a function of his running time t, and the number of parties he corrupts g. Optimizing for t and g gives us the optimal utility of this passive adversary.

Lemma 16 (Optimal utility of a front-running passive adversary, for incentives with limited horizons)**.** *Let* $T_{\mathrm{ub}} > 0$ *be the upper bound on total number of mining queries per round,* $p \in (0,1)$ *be the probability of success of each mining query, and* cutOff $= \omega(\log(\kappa))$ *be the consistency parameter. Given parameters such that* u_{buy} *satisfies limited horizons and protocol* $\Pi^{\not\Bbbk}$*, for* $\mathcal{A}$ *the optimal adversary in* $\mathbb{A}_{fr}$*,* $u_{buy}(\Pi^{\not\Bbbk}, \mathcal{A}) \leq u^{\mathsf{h}}_{\mathsf{honest}} + negl(\kappa)$ *and* $u_{buy}(\Pi^{\not\Bbbk}, \mathcal{A}) + negl(\kappa) \geq u^{\mathsf{l}}_{\mathsf{honest}}$

This lemma directly implies that any utility with limited horizons also has bounded incentives.

Proof. We show the proof for the upper bound. The lower bound can be proven in the exact same way by constructing an optimal front-running, passive adversary for u^{l}_{buy}, except replacing $\mathtt{breward}'(x)$ with $\mathtt{breward}''(x) := \mathtt{breward}(x + t_{ub})$.

Let $\mathtt{breward}'(x) = \mathtt{breward}(x + t_{lb})$. By Lemma 14, we can prove the upper bound on utility in the lemma by constructing an optimal front-running, passive adversary $\mathcal{A}$ for the utility function u^{h}_{buy}.

We first show that, at any round, the optimal passive adversary $\mathcal{A}$ for u^{h}_{buy} is the one who either mines with all his corrupted parties, or does not mine at all.

Claim (1) There is a round t where it is optimal for a passive adversary to do the following: For all rounds $x \leq t$, the adversary does not de-register any party. That is, he mines with all parties corrupted at round t. At round $t+1$, the adversary de-registers all his corrupted parties and stops execution.

Proof of Claim (1): Let g_x be the number of corrupted parties at round x in a given protocol execution. Suppose the adversary mines (queries the random oracle with blocks) with $g \leq g_x$ corrupted parties. Then the payoff for mining at round x is (up to negligible difference)

$$\begin{aligned}\sum_{b \in \mathbb{N}} b \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,x}] - \sum_{q \in \mathbb{N}^2} q \cdot \mathtt{mcost} \cdot \Pr[W^{\mathtt{A}}_{q,x}] \\ = g \cdot p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - g \cdot \mathtt{mcost} \\ = g \cdot (p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost})\,.\end{aligned}$$

Since we assume cutOff $= \omega(\log(\kappa))$ and the adversary is passive, the probability and thus payoff for a fork is negligible. The first equality holds since when the adversary is front-running and passive, all successfully-mined blocks will be added to the ledger. Thus, both $B^{\mathtt{A}}_{b,x}$ and $W^{\mathtt{A}}_{q,x}$ only depend on the number of queries made at round x.

We see that if $(p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost}) \geq 0$ he gains the optimal utility by mining with all his rigs. On the other hand, when $(p \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} - \mathtt{mcost}) < 0$, then he obtains optimal utility by not mining at all. Thus, he does not lose utility by de-registering his parties. There exists a round t described in the claim by assumption of the utility function parameters satisfying bounded mining incentives. ■

Our second claim says that the optimal adversary is the one who statically corrupts parties at the first round. That is, adaptive corruption does not increase his payoffs.

Claim (2) Let G be the total number of parties corrupted by an adversary in a protocol execution and P_G be the associated distribution. Then, (1) given any front-running, passive adversary where $g = \max Supp(G)$, an environment $\mathcal{Z}$ which spawns $g+1$ parties gives the optimal payoff. Moreover, (2) given an environment which spawns at least $g+1$ parties at the first round, the optimal front-running, passive adversary who corrupts at most a total of g parties, is to statically corrupt them at the first round.

Proof of Claim (2): The first statement (1) is given by the fact that the payoff of the passive adversary only depends on the number of successful blocks inserted into the ledger. Since the adversary is front-running, it does not matter how many honest parties there are, as long as the environment spawns enough parties to allow for at least one honest party (for the technical reason that the ledger state is accepted by some honest party). The second statement follows from the fact that Claim (1) implies the adversary's mining strategy at any round does not depend on the number of corrupt parties nor view of the protocol execution. Thus, an adversary corrupting a party at some round x always gains equal or better payoff by corrupting the party at the first round. It is possible for the adversary to corrupt statically first round since we assumed the environment spawns enough parties. ■

Finally, we show that the optimal front-running, passive adversary $\mathcal{A}$ for $u^h_{\mathtt{buy}}$ will corrupt a deterministic number of parties at the first round. By Claims 1 and 2, it suffices to analyze the utility of a front-running, passive adversary $\mathcal{A}$ who corrupts some g parties statically at the first round and mines for t rounds.

$$
\begin{aligned}
&u_{\mathtt{buy}}(\Pi^{\mathtt{B}}, \mathcal{A}) \leq u^h_{\mathtt{buy}}(\Pi^{\mathtt{B}}, \mathcal{A}) \\
&= \sum_{b\in\mathbb{N}} \sum_{x\geq 1} \left(b \cdot \mathtt{breward}'(x) \cdot \mathtt{CR} \cdot \Pr[B^{\mathtt{A}}_{b,x}]\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \sum_{b\in\mathbb{N}} \left(b \cdot \Pr[B^{\mathtt{A}}_{b,1}] \cdot \sum_{x=1}^{t} \mathtt{breward}'(x)\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \left(\sum_{x=1}^{t} \mathtt{breward}'(x)\right) \left(\sum_{b\in\mathbb{N}} b \cdot \Pr[B^{\mathtt{A}}_{b,1}]\right) - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= \mathtt{CR} \left(\sum_{x=1}^{t} \mathtt{breward}'(x)\right) \cdot g \cdot p - g \cdot t \cdot \mathtt{mcost} - \mathtt{ccost}(g) + \mathsf{negl}(\kappa) \\
&= g \cdot \sum_{x=1}^{t} \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) - \mathtt{mcost}\right) - \mathtt{ccost}(g) + \mathsf{negl}(\kappa)
\end{aligned}
$$

To go from line 2 to line 3: for $x \in [1,t]$, $\Pr[B^{\mathtt{A}}_{b,x}] = \Pr[B^{\mathtt{A}}_{b,x'}] = \Pr[B^{\mathtt{A}}_{b,1}]$, and for $x > t$, $\Pr[B^{\mathtt{A}}_{b,x}] = 0$. To go from line 4 to line 5: we see $\sum_{b\in\mathbb{N}} b \cdot \Pr[B^{\mathtt{A}}_{b,1}]$ is the expected number of blocks made in each round, which is gp. We next find the g, t that maximize the above expression. We see that from Claim 1, and that $u_{\mathtt{buy}}$ satisfies limited horizons:

$$\begin{aligned} t &= \arg\max_{t\in\mathbb{N}} \left(\sum_{x=1}^{t} (p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) - \mathtt{mcost}) \right) \\ &= \arg\max_{x\geq 1} (p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) \geq \mathtt{mcost}) \end{aligned}$$

Let m be the mining profit defined by

$$m = \sum_{x=1}^{t} (p \cdot \mathtt{CR} \cdot \mathtt{breward}'(x) - \mathtt{mcost})$$

Then the optimal number of corruptions g is the following (recall T_{ub} is the maximum hashing power per round).

$$g = \arg \max_{g\in[0,\mathsf{T}_{\mathrm{ub}}], g\in\mathbb{N}} (mg - \mathtt{ccost}(g))$$

□

5.4 Analyzing 51% attacks

We can now utilize our above framework to analyze one of the most common types of forking attacks, known as 51% *double-spending* attack [4]. We analyze a range of parameters for utility functions with limited horizons, for which a 51% double-spending adversary breaks the strong attack-payoff security of protocol $\Pi^{\mathbb{B}}$ (formalized by Theorem 14). In more detail, first we will show a general lemma relating the number of honest/adversarial hashes per round, to the time it takes to fork with a 51% double-spending attack (Lemma 17). Then, in Theorem 14 we will show that if the payoff for a successful attack (`fpayoff`) satisfies certain conditions, then an adversary performing a 51% double-spending attack achieves better utility than any passive-mining strategy. This `fpayoff` is quantified as a function of the parameters of the protocol and the utility function.

We call the following strategy a *51% double-spending attack*: The adversary obtains any majority fraction ("51%" is just a colloquial name) of the hashing power, and uses it to secretly mine an extension of the currently longest chain (i.e., keeping successful blocks private to himself), and which he will release after some time. We say that a 51% double-spending attack is *successful* if, when released, the adversary's secret chain is at least as long as the honest chain, and causes

the ledger state of some honest party to fork (which in reality corresponds to a roll-back of more than cutOff blocks, in order to adopt the released attack chain). If this happens, some transactions on the reversed blockchain ledger state may become orphaned (no longer part of the ledger state), thus allowing the attacker to double-spend his coins.

5.4.1 Time to fork

We start by showing a general lemma that relates the amount of honest and adversarial hashing power in a system, to the time to cause a fork via a 51% double-spending attack. That is, how long it takes for an adversary with majority hashing power to secretly create a chain that, when released, would cause an honest party to roll back, or discard, more than cutOff blocks of his own chain in order to adopt the new one.

Definition 32. We say that an adversary $\mathcal{A}$ *causes a fork* in a protocol Π if, except with negligible probability in κ, all simulators $\mathrm{S}^{\mathcal{A}} \in \mathcal{C}_{\mathcal{A}}$ (i.e. those which in fact simulate $\mathcal{A}$ according to UC emulation) use the FORK command[6].

The FORK command, which allows forking the confirmed ledger state (and hence corresponds to rolling back more than cutOff blocks in the real world world), is necessary and sufficient to simulate an adversary who succeeds in a 51% double-spending attack. We compute the (upper bound) time for a 51% double-spending adversary to fork, which is obtained by the time for honest parties to grow their chain by cutOff blocks (for which we can use guaranteed chain-growth of Nakamoto-style blockchains. Since the adversary has more hashing power (and thus more random oracle queries that can be issued sequentially) than the honest party, and since we assume cutOff $= \omega(\log(\kappa))$ and that the adversary does not interfere with the honest parties' mining, this implies that the adversary's secretly-mined chain will be longer than the honest parties' chain, and be the only source for a large rollback, with overwhelming probability in κ.

Lemma 17 (Time to fork with 51% attack)**.** *Let* cutOff $= \omega(\log(\kappa))$*,* $[r, r+t]$ *be any time interval (starting from some round* $r \geq 0$*) of* $t \geq 1$ *rounds,* $\Delta \geq 1$ *be the network delay,* $p \in (0,1)$ *the success probability of one mining query.*

Then for all $\delta, \delta' \in (0,1)$*,* $\alpha \geq \frac{1+\delta}{1-\delta}$*, and* $q \geq 1$ *such that* $t \geq t^{\Delta}_{\delta'}(q)$ *(Definition 30) the following holds. Suppose in time interval* $[r, r+t]$*, (1) the honest parties make at least* q *mining queries per round, and (2) in total they make at most* q_t *queries. Then, the adversary* $\mathcal{A}$ *who performs a 51% double-spending attack for at least* αq_t *queries during the time interval and then releases his secretly-mined chain, causes a fork in the protocol* $\Pi^{\not\Bbbk}$*.* [7]

Proof. As usual we analyze in the state exchange functionality ($\mathcal{F}_{\mathrm{STX}}$) hybrid-world, which ignores negligible-probability bad event like hash collisions that break the tree structure of chains (and in any case can only serve to help the adversary as the honest parties do not take advantage of such bad events).

[6] If $\mathcal{C}_{\mathcal{A}} = \emptyset$, then in any case by definition the utility of $\mathcal{A}$ is infinite.

[7] More concretely, he succeeds except with probability at most $\exp\left(-\frac{\delta^2 \alpha \mu}{2+\delta}\right) + \exp\left(-\frac{\delta^2 \mu}{2}\right) + \exp\left(-\frac{\delta'^2 t \gamma}{2+\delta}\right)$.

Let the adversary's strategy be the following: He does nothing before some round r. Let $\mathcal{C}$ be the longest known chain at the beginning of round r (since the adversary knows all the chains, he knows the longest chain). During $[r, r+t]$, the adversary mines secretly (i.e., mines without releasing any successful blocks), starting from $\mathcal{C}$ (for αq_t queries). After mining in round $r+t$, the adversary releases his secretly-mined chain.

Let c_a be the random variable (r.v.) that is the number of blocks made by the adversary's secret mining, and c_h the r.v. that is the length of the longest chain fragment made by honest parties, during $[r, r+t]$.

First we claim that the adversary causes a fork if, except with negligible probability in κ, the following conditions hold:

(1) $c_a \geq c_h$ and (2) $c_h >$ cutOff.

By (1) and the fact $\mathcal{C}$ is the longest chain at round r, the adversary's secret chain is at least as long as the longest honest chain, so his chain will be adopted by honest parties when he releases it. In addition, by (2), the honest parties must roll back more than cutOff blocks to adopt the adversary's secret chain after it is released, which breaks consistency. In this case, any simulator successfully simulating $\mathcal{A}$ must use the command $\textsc{fork}$.

Let b_h and b_a be the total number of blocks made by the honest and corrupt parties during the time interval. Then $c_h \leq b_h$ and $E(c_h) \leq E(b_h) = \mu = q_t p$. Moreover $c_a = b_a$ since the adversary does mine his own blocks sequentially, and $E(c_a) = E(b_a) = \alpha\mu$.

We recall the Chain Growth Lemma [17, 66, 115] that relates the time it takes an honest chain to grow by some T blocks with the honest parties' hashing power and the network delay. We are in particular interested in $T =$ cutOff—that is, how long it takes for the honest chain to grow by cutOff blocks. This lemma tells us that if $t \geq \frac{\mathsf{cutOff}}{(1-\delta)\gamma}$ (implied by our assumption) then $c_h \geq$ cutOff except with probability at most $\mathsf{negl}(\gamma t)$. Since cutOff $= \omega(\log(\kappa))$, we have $\mathsf{negl}(\gamma t) = \mathsf{negl}(\kappa)$.

We analyze below the probability that either condition (1) or (2) above fail. To get from line 1 to line 2: By $\alpha \geq \frac{1+\delta}{1-\delta}$, we have $\alpha q p_t(1-\delta) \geq q_t p(1+\delta)$; moreover $c_h \leq b_h$. To get line 3 to line 4: We apply Chernoff bound and Chain Growth Lemma.

$$
\begin{aligned}
&\Pr\left[(c_a < c_h) \vee (c_h \leq \mathsf{cutOff})\right] \\
&\leq \Pr\left[(c_a \leq \alpha q_t p(1-\delta)) \vee (b_h \geq q_t p(1+\delta)) \vee (c_h \leq \mathsf{cutOff})\right] \\
&\leq \Pr\left[c_a \leq \alpha q_t p(1-\delta)\right] + \Pr\left[b_h \geq q_t p(1+\delta)\right] + \Pr\left[c_h \leq \mathsf{cutOff}\right] \\
&\leq \exp\left(-\frac{\delta^2 \alpha \mu}{2+\delta}\right) + \exp\left(-\frac{\delta^2 \mu}{2}\right) + \exp\left(-\frac{\delta'^2 t \gamma}{2+\delta}\right)
\end{aligned}
$$

where $\mu = q_t p$. Since we assume $\mu = q_t p \geq w = \omega(\log(\kappa))$, $\exp(-c\mu) = \exp(-\omega(\log(\kappa)))$ is negligible in κ for any constant c.[8]

□

[8] To prove that it is negligible, for contradiction suppose there is a polynomial x^d that is asymptotically larger than e^g for some $g = \omega(\log(\kappa))$. But $x^d = e^{d\log(x)}$ which implies $g = O(\log(x))$ which is a contradiction.

A visualization. In Figure 5.1, the (upper-bound) time to fork with exactly 51% corruption, is graphed against the total number of rigs in the system. The graph uses the formula from Lemma 17. We use current parameters for Ethereum Classic as the source of the concrete parameters for this figure, and refer the reader to Sec. 5.6 for more details.

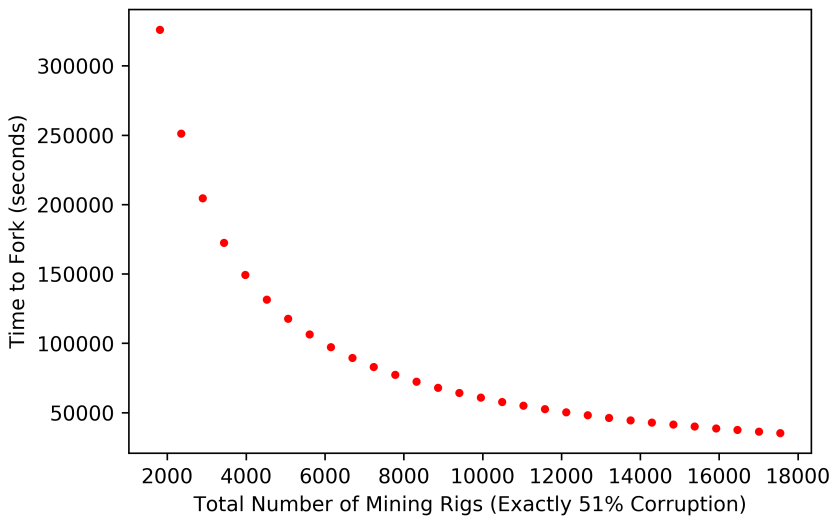

Figure 5.1: Time to create a fork via 51% attack, versus the total number of mining rigs. Here the adversary corrupts exactly 51%.

5.4.2 Payoff of 51% double-spending attacks

In this section, we prove Theorem 14 and its corollary Theorem 15, which quantify the size of the payoff for double-spending, under which a 51% double-spending attack can break strong attack-payoff security. That is, the attacker achieves better utility than *any* passive mining strategy. While one may think that it is always profitable to attack if there is no assumption on the honest majority of hashing power, there are a few things that may deter an attacker. For example, the costs of buying or renting mining equipment for the attack may become too high compared to constant double-spending payoff and the diminished block rewards as time goes on. Our statement below quantifies an amount of payoff for forking (e.g. how much an attacker can double-spend) to incentivize a 51% double-spending attack. Intuitively, the result below says that as long as the payoff for forking (`fpayoff`) is larger than the loss of utility from withholding blocks and corrupting a large number of parties to perform the attack, then there is a 51% attack strategy that is more profitable than *any* front-running, passive adversary.

Theorem 14 (51% double-spending attacks that break (strong) attack-payoff security (u_{buy}))**.** *Let* $T_{\text{ub}} > 2$ *be the upper bound on total number of mining queries per round,* $p \in (0,1)$ *be the probability of success of each mining query, and* cutOff $= \omega(\log(\kappa))$ *be the consistency*

parameter. Then, the protocol $\Pi^{\mathbb{B}}$ *is not attack-payoff secure/strongly attack-payoff secure in any attack model* $\mathcal{M}$ *whose induced utility function* u_{buy} *satisfies limited horizons, if for some* $\delta \in (0,1)$, $\alpha > 1$ *and* $g = \frac{T_{\text{ub}}}{1+\alpha}$ *the following holds:*

$$\mathit{fpayoff} > u^{\mathsf{h}}_{\mathsf{honest}} - \alpha \cdot g \cdot t^{\Delta}_{\delta}(g) \left(p \cdot \mathit{CR} \cdot \mathit{breward}(t^{\Delta}_{\delta}(g) + t_{ub}) - \mathit{mcost}\right) - \mathit{ccost}(\alpha g).$$

Proof. First we show an upper bound on the utility of an optimal passive adversary $\mathcal{A}_1$. Then, we show that there is a 51% attacking adversary $\mathcal{A}_2$ who achieves better utility than $\mathcal{A}_1$.

We show the former, by showing the optimal payoff of any pair $(\mathcal{Z}, \mathcal{A}_1)$, where $\mathcal{Z}$ is the environment and $\mathcal{A}_1$ a front-running and passive adversary in $\Pi^{\mathbb{B}}$. This is directly from Lemma 16, which shows that $u_{\mathsf{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}_1) \leq u^{\mathsf{h}}_{\mathsf{honest}} + \mathsf{negl}(\kappa)$ for any utility function u_{buy} that satisfies limited horizons.

Then, we show that there is an environment under which an (malicious) adversary $\mathcal{A}_2$ obtains non-negligibly more utility than $\mathcal{A}_1$. Consider the adversary that corrupts $\alpha g = \frac{\alpha \mathsf{T}_{\text{ub}}}{1+\alpha}$ parties, and mines a secret chain for $t^{\Delta}_{\delta}(g)$ rounds (so he will make $\alpha \cdot g \cdot t^{\Delta}_{\delta}(g)$ mining queries) before releasing his secret chain and then deregisters his corrupted parties. By Lemma 17, he successfully causes a fork with overwhelming probability in κ. However, by withholding blocks (in order to create a fork), he also loses some block rewards. This is because his blocks would end up in an honest party's ledger at a later time than if he just broadcasted successful blocks immediately after creating them. Consider an environment which spawns T_{ub} parties (which are honest as the adversary does not corrupt them) after he releases his secret chain, which will reduce the time between when the adversary broadcasts his secret chain and when the chain becomes confirmed as part of the ledger to at most t_{ub} (Lem. 14). This means the adversary $\mathcal{A}_2$ achieves utility of:

$$u_{\mathsf{buy}}(\Pi^{\mathbb{B}}, \mathcal{A}_2) \geq \alpha \cdot g \cdot t^{\Delta}_{\delta}(g) \left(p \cdot \mathtt{CR} \cdot \mathtt{breward}(t^{\Delta}_{\delta}(g) + t_{ub}) - \mathtt{mcost}\right) - \mathtt{ccost}(\alpha g) + \mathtt{fpayoff}.$$

We ignore negligible probability/utility loss from an unsuccessful attack. We also recall that t_{ub} is the upper bound from Lemma 14 on the time between a block is created, and the block enters a honest party's ledger (we also recall the utility is computed using the optimal environment for the adversary). Since $\mathtt{breward}(\cdot)$ is a non-increasing function, this lower-bounds the payoff the adversary obtains from creating blocks. Given the conditions on $\mathtt{fpayoff}$ it then holds that $\mathcal{A}_2$ breaks strong attack-payoff security. Because the attack provokes the fork-command, also attack-payoff security cannot hold.

□

We state the case where the adversary mines with rented equipment (and uses utility function $u_{\mathtt{rent}}$), as a direct corollary to Theorem 14.

Theorem 15 (51% double-spending attacks that break (strong) attack-payoff security ($u_{\mathtt{rent}}$))**.** *Let* $T_{\text{ub}} > 2$ *be the upper bound on total number of mining queries per round,* $p \in (0,1)$ *be the probability of success of each mining query, and* cutOff $= \omega(\log(\kappa))$ *be the consistency*

parameter. Then, the protocol $\Pi^{\not\Bbbk}$ *is not attack-payoff secure/strongly attack-payoff secure in any attack model* $\mathcal{M}$ *whose induced utility function* u_{rent} *satisfies limited horizons, if for any* $\delta \in (0,1)$, $\alpha > 1$ *and* $g = \frac{T_{\text{ub}}}{1+\alpha}$ *the following holds:*

$$\mathit{fpayoff} > u^{\mathsf{h}}_{\text{honest}} - \alpha \cdot g \cdot t^{\Delta}_{\delta}(g)\left(p \cdot \mathit{CR} \cdot \mathit{breward}(t^{\Delta}_{\delta}(g) + t_{ub}) - \mathit{mcost}\right).$$

5.4.3 Visualizations with concrete values

We will visualize Theorems 14 and 15 through Figures 5.2 and 5.3. We consider two utility functions, one where the adversary buys mining equipment, and one where the adversary rents. We then graph the utilities of passive/non-passive adversaries, against the maximum fraction of corrupted parties. The concrete parameters are based on current (as of writing, Feb. 2021) parameters for Ethereum Classic. The outline is given in Sec. 5.6.

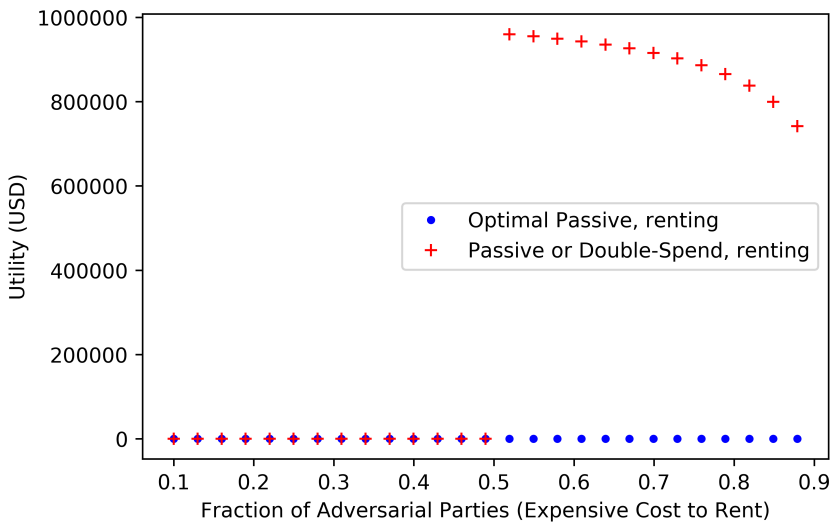

Figure 5.2: Utility of the passive/51% double-spending attacker who rents hashing power, versus the fraction of adversarial parties. Here we consider an expensive cost to rent hashing power (1.96 BTC/TH/day, at $\$50,000$/BTC).

In Figure 5.2, we consider the incentives of a 51% attacker who rents his hashing power, using the price for renting of 1.96 BTC/TH/day (Bitcoin per terahash per day), at $\$50,000$/BTC. In this case, it is in fact not profitable to mine passively (and thus the optimal passive strategy is to not mine at all). However, when the adversary corrupts more than majority of hashing power, it may become profitable to mine in order to create a fork. It is less profitable for the adversary to

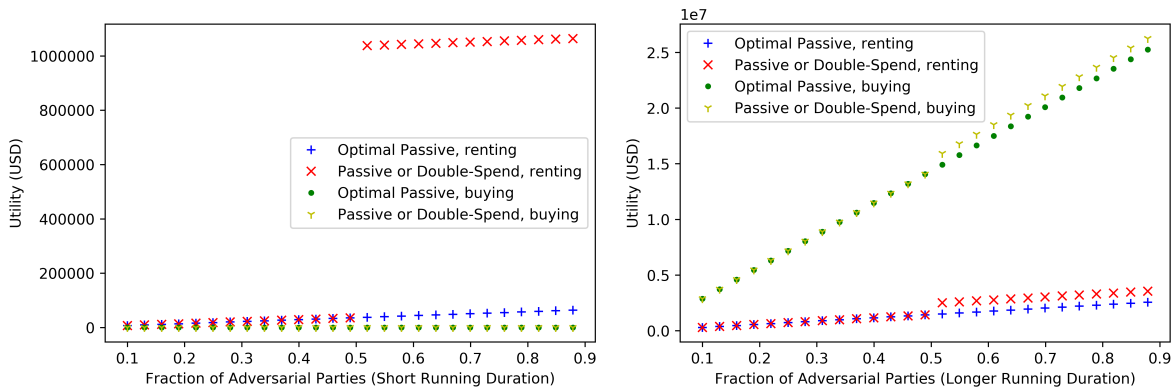

Figure 5.3: Utility of the passive/51% double-spending attacker versus the fraction of adversarial parties. We consider an attacker who runs for a short duration (1 week) and a long duration (40 weeks).

corrupt a larger fraction of the parties, as cost of renting becomes too high. We remark that even when it is not profitable to mine (passively) using rented rigs, this does not exclude incentivizing honest parties from mining with e.g., bought equipment.

In the next two examples in Figure 5.3, we compare the utility of the attacker who mines with purchased rigs, and one who mines with rented rigs. For the attack who buys hashing power, each rig costs \$3000, and mining (electricity) costs \$0.000047/s. For the attacker who rents, for more interesting comparisons we consider a cheaper cost to rent hashing power (1.96 BTC/TH/day, at a cheaper $\$22,000$/BTC). We consider two scenarios: the attacker either (1) only plans to mine for a short duration of one week, or (2) plans to mine for a longer duration of 40 weeks (time is expressed in seconds in the code). For the purposes of the graphs, we account for the possible variance of Bitcoin-USD exchange rates by using an average exchange rate over the relevant period of time. In either case, to more closely model reality, we restrict the duration of the attack, where the adversary may obtain a majority of hashing power, to 3 days (which, in the code, simply means we do not show attacks that last longer than 3 days). We see a big difference between the two scenarios. In the short duration case, it is much more profitable to mine or attack with rented rigs. In fact, it is not even profitable to fork using purchased rigs, as the cost of purchase is higher than the payoff for double-spending. The long duration case is the opposite. Although it may be profitable to mine in both cases, it is vastly more profitable to mine and attack with purchased rigs than rented rigs. This agrees with our intuition and reality: the high initial investment of buying mining equipment is offset in the long run by the lower cost of mining. Moreover, an attacker who is only interested in mining in order to perform a 51% attack for a short time is incentivized to use hash renting services.

5.5 Mitigating 51% attacks

In previous sections, we studied utility functions with limited horizons, in which an attacker is incentivized to perform a 51% double-spending attack and break (strong) attack-payoff security. In this section, we turn to analyzing how to *defend* against 51% attacks. Specifically, given an attacker's utility function with limited horizons, and a cut-off parameter cutOff that achieves security in the honest majority setting, we show a way to amplify cutOff to obtain security against a rational (and possibly dishonest majority) attacker.

To show attack payoff security, one must show that for *any* adversarial strategy attacking the protocol, there is another adversary who attacks the dummy protocol with access to the ideal ledger functionality $\mathcal{G}_{\text{LEDGER}}$[9], which achieves the same or better utility.

Even more difficult, we place very few restrictions on the adversary: he may corrupt any fraction of parties (e.g. more than majority) and perform any currently known (e.g. block withholding) or unknown strategy. The only restriction we place on the attacker is that he is incentive-driven. Fortunately, a rational attacker is limited by his utility function. As we show, given a utility function satisfying limited horizon, we are able to bound the amount of mining an incentive-driven adversary will do, even in presence of a payoff for forking. Then, by choosing a large enough consistency parameter cutOff, we ensure that attackers are disincentivized from creating a fork.

More specifically: We first present in Section 5.5.1 a result that shows that if an adversary's hashing resources are limited by a budget B, then there is a bound on the interval of rounds where the blockchain is at risk of a consistency failure (Lemma 18). For this, we apply a result from [14] that, roughly, shows how fast a blockchain's consistency can recover after an attack by an adversary with a given budget (the self-healing property of Bitcoin). Based on this fundamental property, we present in Section 5.5.2, the main result of the section: Given a utility function with limited horizons, we show a condition on the parameter cutOff, depending only on the utility function and protocol parameters, such that $\Pi^{\text{฿}}$ is attack-payoff secure. To do so, we show that an adversary who spends too much budget will begin to lose utility (Lemma 19), and then combine this result with that of Section 5.5.1.

5.5.1 Budget to vulnerability period

Assume an instance of the Bitcoin backbone protocol with cut-off parameter ℓ. We distinguish here between ℓ and cutOff for clarity, since we will eventually amplify ℓ to obtain our final cut-off parameter cutOff.

Under the honest majority condition, we know that a consistency failure (expressed as the probability that blocks which are ℓ deep in an honest parties adopted chain can be reverted) appears with probability negligible in ℓ (and consequently also in any security parameter κ as long as $\ell = \omega(\log(\kappa))$. We now recall (and state a simple corollary from) the result from [14], which

[9] Recall this is the ledger functionality that ensures consistency and liveness, but since we will amplify the cut-off parameter cutOff, we only achieve worse overall chain growth and chain quality parameters for the ledger (cf. [14, 17]).

defines a relationship between an adversary's violation of honest-majority (measured as a so-called budget B by which it can violate the honest-majority condition) and the time until Bitcoin (and more generally, Nakamoto-style PoW chains) self-heals after the adversary returns to below 50% hashing power. That is, until Bitcoin can again guarantee consistency for the part of the chain that is at least ℓ blocks deep in a longest chain held by an honest party. The self-healing time depends on the budget and the parameter ℓ. Recall that θ_{pow} is the usual security threshold for Bitcoin as explained in Section 2.5.

Definition 33 ($(\theta_{pow}, \epsilon, \mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B)$-adversary [14])**.** Let $\theta_{pow}, \epsilon \in (0,1)$, $\mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B \in \mathbb{N}$. A $(\theta_{pow}, \epsilon, \mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B)$-adversary is an adversary[10] satisfying the following: At every round i, let n_a^i and n_h^i be the mining queries made by corrupt and honest parties in this round. Then, (1) For all i, $\mathsf{T}_{\mathrm{lb}} \leq n_a^i + n_h^i \leq \mathsf{T}_{\mathrm{ub}}$, and (2) For all i, $n_a^i \leq (1-\epsilon) \cdot \theta_{pow} \cdot n_h^i + B_i$, where $B_i \geq 0$ and $\sum_i B_i = B$.

We say the adversary *attacks between rounds* $a < b$ if $B_i = 0$ for any $i < a$ or $i > b$ (i.e. he spends all his budget between rounds a and b).

We say an adversary *spends budget* B *over* t *rounds*, if the adversary has budget B, and only spends it in rounds $r_1 < r_2 < \cdots < r_t$, such that $\sum_i B_{r_i} = B$.

The behavior of a blockchain protocol under an attack by an $(\theta_{pow}, \epsilon, \mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B)$-adversary is described by a *vulnerability period*. The vulnerability period is an upper bound on number of rounds before and after an adversary performs the attack, such that protocol is still at risk of a (non-negligible) consistency failure.

Definition 34 (Consistency self-healing property and vulnerability period [14])**.** A protocol is self-healing with vulnerability period (τ_l, τ_h) with respect to consistency, and against a $(\theta_{pow}, \epsilon, \mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B)$-adversary who attacks between rounds (a, b), if the consistency failure event $\mathsf{ConsFail}_\ell(r)$ occurs except with at most negligible probability unless $r \in [\rho_\alpha - \tau_l, \rho_\beta + \tau_h]$. $\mathsf{ConsFail}_\ell(r)$ is defined as the event that ℓ-consistency is violated in an execution for rounds (r, r'), w.r.t. some round $r' > r$, and any two pairs of honest parties.

In other words, outside of these "dangerous" rounds $[a - \tau_l, b + \tau_h]$, chains adopted by honest parties are guaranteed to diverge by at the most recent ℓ blocks. Below, [14] gives a characterization of the vulnerability period in terms of the budget B.

Theorem 16 (From [14])**.** *A Nakamoto-style PoW blockchain with an upper bound* T_{ub} *of hashing queries per round, maximum network delay* Δ*, success probability* p*, and cut-off parameter* ℓ *satisfies the* consistency self-healing property *with vulnerability period* $(\tau_l, \tau_h) = (O(B), O(B) + O(\ell))$ *against any* $(\theta_{pow}, \epsilon, \mathsf{T}_{\mathrm{lb}}, \mathsf{T}_{\mathrm{ub}}, B)$*-adversary, for any* $\epsilon, \mathsf{T}_{\mathrm{lb}} > 0$.

The vulnerability period only bounds the number of "bad" rounds before the attack, and after *all* the budget is spent. For our treatment, we consider a more applicable version of the vulnerability period. In Lemma 18, we show the maximum number of consecutive rounds where ConsFail may occur, by applying the above theorem in a piece-wise fashion. For example, if the

[10] Here the environment is also included in this statement.

adversary spends his budget over a long period of time (e.g., spend a bit of the budget, wait for 2 years, then spend more of his budget), the theorem is not directly suitable for our needs, but it is possible to isolate those "spending" rounds and applying the theorem to each such region. Then, since the total hashing power in the system is bounded, we can use this maximum consecutive "bad" rounds to bound the maximum number of blocks that can be rolled back at any given round.

Lemma 18 (Max consecutive consistency failure rounds and associated number of blocks and rollback)**.** *In the same setting as above in Theorem 16, except with negligible probability the following holds: for any adversary with budget B, spent over t rounds (that is, for t different rounds i it holds that $B_i > 0$), there is a maximum number $R(B,t,\ell) = O(B) + O(\ell t)$ of consecutive rounds r_j where* $\mathsf{ConsFail}_\ell(r_j)$ *occurs, during which at most $W(B,t,\ell) = 2\mathsf{T}_{\mathrm{ub}}p \cdot R(B,t,\ell)$ blocks are created.*

Proof. To show $R(B,t,\ell)$ we note that the rounds where $\mathsf{ConsFail}_\ell$ occurs, are either (1) rounds where adversary spends his budget (i.e., when adversary has majority) (2) some number of rounds before some amount of budget is spent, bound by τ_ℓ in Theorem 16 (3) some number of rounds after some amount of budget is spent, bound by τ_h in Theorem 16. For (1), there are exactly t rounds by definition. For (2), we observe that this is at most $O(B)$, regardless of how the adversary spends his budget. For (3), we see that this is at most $O(B) + O(\ell t)$, since τ_h gives the adversary at most $O(\ell)$ rounds (where $\mathsf{ConsFail}_\ell$ may occur) each time the budget is spent, and there are exactly t such rounds. Summing the above, we get $R(B,t,\ell) = O(B) + O(\ell t)$. We obtain $W(B,t,\ell)$ through Chernoff bound (and for simplicity using 2 instead of $(1+\delta)$), given that there are at most T_{ub} hashes per round.

□

Looking ahead, this means that at any point in time, prefixes of honest parties' chains must agree (except with negligible probability) when dropping the most recent $W(B,t,\ell) + \ell$ blocks. Here, we omit the dependency on p because we treat it as a constant parameter of the protocol.

5.5.2 Attack-payoff security

In this section we will show the following: For any utility function with limited horizons, we give a characterization of how to adjust the consistency parameter (depending on the protocol parameters and those of the utility function) such that $\Pi^{\bitcoin}$ is attack payoff secure. To do so, we will first upper bound the utility of adversaries who spends a total budget B over some time t, given a utility function $u_{\mathtt{buy}}$ with limited horizons (Lemma 19, and Corollary 3 for utility function $u_{\mathtt{rent}}$). In Theorem 17, we then combine this lemma with the result of the previous subsection, and present our characterization of parameters for which $\Pi^{\bitcoin}$ is attack-payoff secure—i.e. for which forks are disincentivized.

Below, we quantify an upper bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ on the utility of *any* adversary spending budget of at least B over exactly t rounds, assuming the utility function satisfies limited horizons. Why are we interested in this quantity? Recall $W(B,t)$—which informally represents an interval of blocks where consistency might fail—increases with B and t. Looking ahead, we will find a large

enough $W(B,t)$ that disincentivizes attacks (i.e., $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) < 0$). To show that the upper-bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ is useful, later we will show that it is possible use it to derive a maximum B,t, which we denote by $\bar{B},\bar{t}$.

Lemma 19 (Upper bound utility of adversary spending budget at least B, over time t)**.** *Suppose $u_{\mathtt{buy}}(\Pi^{\mathtt{Ƀ}},\mathcal{A})$ satisfies limited horizons. Then an adversary $\mathcal{A}$ with budget at least $B > 0$, and who spends it over exactly $t \geq \frac{B}{\mathsf{T}_{\mathrm{ub}}-\bar{n}_a}$ rounds, achieves utility at most $u_{\mathtt{buy}}(\Pi^{\mathtt{Ƀ}},\mathcal{A}) \leq u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ where*

$$\begin{aligned} u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) := &\sum_{x=1}^{t_h} \mathsf{T}_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &+ \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &- \mathtt{ccost}\left(\bar{n}_a + \frac{B}{t}\right) + \mathtt{fpayoff} \end{aligned}$$

and where $t_h := \arg\max_{x\in\mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost})$, $\bar{n}_a := \frac{(1-\epsilon)\cdot\theta_{pow}\cdot\mathsf{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

If $t < \frac{B}{\mathsf{T}_{\mathrm{ub}}-\bar{n}_a}$ (in this case it is not possible to spend budget B over t rounds) or $B \leq 0$, then $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ is undefined.

Proof. We first note that if $t < \frac{B}{\mathsf{T}_{\mathrm{ub}}-\bar{n}_a}$ then it is by definition not possible for an adversary to spend at least budget B over t rounds: In each round, at most T_{ub} hash queries can be made, and given a lower bound of T_{lb} mining queries in each round, the adversary must make more than $\bar{n}_a$ queries in order to spend budget.

Thus, let $\mathcal{A}$ be any adversary who spends at least budget B, over exactly t rounds for $t > \frac{B}{\mathsf{T}_{\mathrm{ub}}-\bar{n}_a}$. Let T be the maximum running time of $\mathcal{A}$ in some environment $\mathcal{Z}$. Let $q_1,\cdots,q_T$ be the random variables where q_i is the number of queries $\mathcal{A}$ makes in round i. Let $g = \max_{i\in[1,T]} q_i$ be the maximum number of parties $\mathcal{A}$ corrupts at any given round. Let $t_h = \max_{x\in\mathbb{N}} x : p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost}$, which exists as $u_{\mathtt{buy}}$ satisfies limited horizons. Then,

$$\begin{aligned} u_{\mathtt{buy}}(\Pi^{\mathtt{Ƀ}},\mathcal{A}) &\leq \sum_{x=1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}(g) + \mathtt{fpayoff} \\ &= \sum_{x=1}^{t_h} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \\ &\quad + \sum_{x=t_h+1}^{T} q_i (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}(g) + \mathtt{fpayoff}. \end{aligned}$$

Now by construction $p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq 0$ for $x \leq t_h$ so it is optimal to make as many queries as possible in these rounds. Thus, $\sum_{x=1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \leq \sum_{x=1}^{T} g \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$. On the other hand, $p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) < 0$ for $x > t_h$ so it is optimal to

make as few queries as possible in these rounds. Thus, $\sum_{x=t_h+1}^{T} q_i \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) \leq \sum_{x=t_h+1}^{t} \bar{n}_a \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$. The latter statement is because by assumption that $\mathcal{A}$ spends his budget for t rounds, we have $T \geq t$ and he must make at least $\bar{n}_a + 1$ queries when he spends his budget. Note we are ignoring whether a fork can be obtained this way, as in the upper-bound we give `fpayoff` to the adversary "for free".

In addition, it must be that $g \in [\bar{n}_a + \frac{B}{t}, \mathsf{T}_{\mathrm{ub}}]$—he must corrupt enough parties to spend B budget in t time, but corrupting more than T_{ub} (the upper bound on total hashing power in the system) parties does not increase his hashing power. This gives us the bound in the lemma (due to limited horizons, `ccost` is a non-decreasing function of g).

□

As a corollary, by setting $\mathtt{ccost}(\cdot) = 0$ and $\mathtt{mcost} = \mathtt{rcost}$, we obtain an upper bound on the utility of any adversary who spends at least budget B, assuming the utility function satisfies limited horizons with renting.

Corollary 3. *Suppose $u_{\mathtt{rent}}(\Pi^{\not\Bitcoin}, \mathcal{A})$ satisfies limited horizons with renting. Then an adversary $\mathcal{A}$ who spends budget of at least $B > 0$ over exactly $t \geq \frac{B}{\mathsf{T}_{\mathrm{ub}} - \bar{n}_a}$ rounds, achieves utility at most $u_{\mathtt{rent}}(\Pi^{\not\Bitcoin}, \mathcal{A}) \leq u^{\mathtt{ub}}_{\mathtt{rent}}(B, t)$ where*

$$u^{\mathtt{ub}}_{\mathtt{rent}}(B,t) := \sum_{x=1}^{t_h} \mathsf{T}_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{rcost}) + \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{rcost}) + \mathtt{fpayoff}$$

and where $t_h := \arg\max_{x \in \mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{rcost})$, $\bar{n}_a := \frac{(1-\epsilon)\cdot\theta_{pow}\cdot\mathsf{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

A natural question is whether the upper bound $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t)$ and $u^{\mathtt{ub}}_{\mathtt{rent}}(B,t)$ will be useful for bounding B, t. We remark that it is not even trivially clear whether they bounded, as the budget does not limit how many rounds (when honest majority is satisfied) the adversary can mine. Below, we show that there indeed exist a maximum B, t for which $u^{\mathtt{ub}}_{\mathtt{buy}}(B,t) \geq 0$ (resp. $u^{\mathtt{ub}}_{\mathtt{rent}}(B,t) \geq 0$, which we denote by $\bar{B}, \bar{t}$.

Lemma 20. *$\bar{B} := \arg\max_{B>0}\left(u^{\mathtt{ub}}_{\mathtt{buy}}(B,\cdot) \geq 0\right)$ and $\bar{t} := \arg\max_{t>0}\left(u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) \geq 0\right)$ exist, or $\forall t > 0, u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot,t) < 0$. The same is true when replacing $u^{\mathtt{ub}}_{\mathtt{buy}}$ with $u^{\mathtt{ub}}_{\mathtt{rent}}$ in the statement.*

Proof. We prove for $u^{\mathtt{ub}}_{\mathtt{buy}}$ then the case for $u^{\mathtt{ub}}_{\mathtt{rent}}$ follows. Let $b_{\max} = \sum_{x=1}^{t_h} \mathsf{T}_{\mathrm{ub}} \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost})$ where $t_h = \arg\max_{x \in \mathbb{N}}(p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) \geq \mathtt{mcost})$ (exists by limited horizons), Let $\bar{n}_a = \frac{(1-\epsilon)\cdot\theta_{pow}\cdot\mathsf{T}_{\mathrm{lb}}}{1+(1-\epsilon)\cdot\theta_{pow}}$.

Suppose there is a $t > 0$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0$; we will show that (1) There is a $t \geq t_h$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(B, t) \geq 0$, and (2) $\bar{t} := \arg\max\left(u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0\right)$ exists. For (1),

$$
\begin{aligned}
&u^{\mathtt{ub}}_{\mathtt{buy}}(B, t) \\
&\leq b_{\max} + \sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}\,(\bar{n}_a + 1) + \mathtt{fpayoff} \\
&\leq b_{\max} + \sum_{x=t_h+1}^{t=t_h} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) - \mathtt{ccost}\,(\bar{n}_a + 1) + \mathtt{fpayoff}.
\end{aligned}
$$

The second line is by limited horizons: ccost is a non-decreasing function, so $\mathtt{ccost}\left(\bar{n}_a + \frac{B}{t}\right) > \mathtt{ccost}\,(\bar{n}_a + 1)$. The third line is by assumption $p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost} < 0$ for $x > t_h$, and taking $t = t_h$ makes this term zero. Since there are no other terms with t, and by assumption that there exists $t > 0$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0$, this means $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0$ for some $t \geq t_h$.

For (2), We bound $\bar{t}$ first, then $\bar{B} < \bar{t}\,(\mathsf{T}_{\mathrm{ub}} - \bar{n}_a)$ follows from condition $t < \frac{B}{\mathsf{T}_{\mathrm{ub}} - \bar{n}_a}$. By limited horizons, there is a constant (in the security parameter) $\delta > 0$ such that for all $x \geq t_h + 1$, $\mathtt{mcost} - p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) > \delta$. Thus,

$$
\begin{aligned}
\bar{t} &:= \arg\max_{t>0} \left(u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0\right) \\
&\leq \arg\max_{t>0} \Big(b_{\max} + \mathtt{fpayoff} - \delta \cdot (\bar{n}_a + 1)(t - t_h - 1) - \mathtt{ccost}(\bar{n}_a + 1) \geq 0\Big) \\
&= \Big(b_{\max} + \mathtt{fpayoff} + \delta \cdot (\bar{n}_a + 1)(t_h + 1) - \mathtt{ccost}(\bar{n}_a + 1)\Big) \Big/ \Big(\delta \cdot (\bar{n}_a + 1)\Big).
\end{aligned}
$$

The second line is from minimizing the term $\mathtt{ccost}(\bar{n}_a + B/t)$ as $\mathtt{ccost}(\bar{n}_a + 1)$ (by limited horizons, ccost is a non-decreasing function) and minimizing $\sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (p \cdot \mathtt{CR} \cdot \mathtt{breward}(x) - \mathtt{mcost}) = -\sum_{x=t_h+1}^{t} (\bar{n}_a + 1) \cdot (\mathtt{mcost} - p \cdot \mathtt{CR} \cdot \mathtt{breward}(x))$ as $-\delta \cdot (\bar{n}_a + 1)(t - t_h - 1)$ (given what we showed in (1), we have $t \geq t_h$). The third line is positive since we showed there is a $t > t_h$ such that $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) > 0$.

□

5.5.2.1 Concrete examples of $\bar{B}$, $\bar{t}$.

We give example estimate values of $\bar{t}$, which we recall is the upper bound on the period of time where the attacker is incentivized to spend budget. These values are obtained using the equation from proof of Lemma 20, and from them $\bar{B}$ can be computed easily as $\bar{B} = \bar{t} \cdot (\mathsf{T}_{\mathrm{ub}} - \bar{n}_a)$.

In more detail: In the case there is no t such that $u^{\mathtt{ub}}_{\mathtt{buy}}(\cdot, t) \geq 0$ (resp. $u^{\mathtt{ub}}_{\mathtt{rent}}(\cdot, t) \geq 0$) ; i.e., it is not profitable to perform a 51% attack at all), we display $\bar{t} = 0$. We use $\mathsf{T}_{\mathrm{lb}} \cdot (1 - \epsilon) \approx 1$ to compute parameter $\bar{n}_a$, which requires an attacker to corrupt at least $\sim 50\%$ of the total hashing power. For simplicity we assume that the total hashing power is relatively stable during the period of attack, and thus we set $\mathsf{T}_{\mathrm{lb}} = \mathsf{T}_{\mathrm{ub}}$. We refer to Sec. 5.6 for details on the other parameters used (e.g., electricity costs, cost per mining rig, block rewards, etc.), which are estimates from Ethereum Classic at the time of writing (Feb. 2021).

t_h (days)	$\bar{t}$ (days)
0.33	5.9
0.66	6.3
1	6.6
2	7.7
3	8.7
6	11.9
9	15.1

Table 5.1: $\bar{t}$ varying over anticipated profitable mining time t_h, price/rented rig 0.00023/unit of hashing power ($=$ 435MH), which is \$179,592 per day.

`rcost` (USD)	$\bar{t}$ (days)	Cost/day (USD)
\$0.0001	24.0	\$78,084
\$0.0002	10.5	\$156,167
\$0.0003	4.3	\$234,251
\$0.0004	3.2	\$312,334
\$0.0005	2.6	\$390,418
\$0.0006	2.1	\$468,501

Table 5.2: Estimated $\bar{t}$ and cost of attack per day, varying over price per rented mining rigs per second. If passive-mining is profitable at all, we set $t_h = 3$ days, and otherwise $t_h = 0$ by default.

In the case of bought mining rigs, we see that $\bar{t} = 0$ for $t_h \leq 89$ days (recall that t_h is the length of time an attacker expects passive-mining to be profitable). Then, we estimate $\bar{t} = 96$ days if $t_h = 90$ days, and $\bar{t} = 196$ days if $t_h = 100$ days. However, the estimated cost to buy a majority of mining rigs can be in the order of tens of millions in USD (using our estimated per-rig cost, around \$27 million). This high cost of attack, coupled with the difficulty of purchasing and maintaining such a large number of rigs, may suggest why a 51% attacker might be incentivized to rent mining rigs instead.

In the case of rented mining rigs, we see that the estimated cost of attack becomes more reasonable. Below, we show estimated values of $\bar{t}$, varying both over t_h (Table 5.1) and the price of renting `rcost` (Table 5.2, Fig. 5.4). In particular, we see that for a realistic renting price of \$0.00045/unit of hashing power, the estimated $\bar{t}$ is around 3 days—on par with the 2-day interval of attack on Ethereum Classic, which we base our numbers on, in August, 2020.

Finally, we can make a general statement about the attack payoff security of protocol $\Pi^{฿}$ for any utility function satisfying limited horizons. Informally: our utility function limits how much budget B (spent over how many rounds t) any incentive-driven attacker could reasonably have. Then, if the the window size is large enough to accommodate the largest of such budgets, the protocol is attack-payoff secure. In the following, $u^{\mathtt{ub}}_{\mathsf{buy}}(B,t)$, $u^{\mathsf{I}}_{\mathsf{honest}}$, and $W(B,t)$ are from Lem. 19, Lem. 16, and Lem. 18 respectively. The equivalent statement for utility function $u_{\mathtt{rent}}$ can be obtained by using the corollaries of the results above for the case of renting.

Theorem 17. *Let $T_{\mathrm{ub}}, T_{\mathrm{lb}} > 0$ be the upper and lower bounds on total number of mining queries per round and $p \in (0,1)$ be the probability of success of each mining query and let $\ell = \omega(\log(\kappa))$. Then, $\Pi^{฿}$ with consistency parameter* cutOff *is attack-payoff secure in any model*

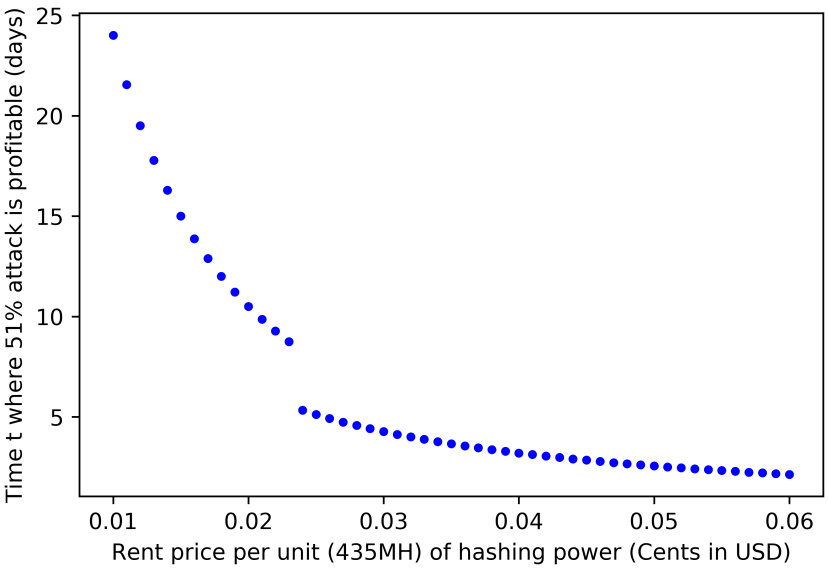

Figure 5.4: Graph representation of Table 5.2.

$\mathcal{M}$, whose induced utility u_{buy} satisfies limited horizons, whenever the condition holds that

$$\text{cutOff} \quad > \quad \ell \; + \max_{(B,t):u^{\text{ub}}_{\text{buy}}(B,t)>u^{\text{l}}_{\text{honest}}} W(B,t,\ell).$$

The same statement is true replacing u_{buy} with u_{rent} and u^{ub}_{buy} with u^{ub}_{rent}.

Proof. We prove for u_{buy} and $u^{\text{ub}}_{\text{buy}}$, and the renting case follows. We showed that for any (B,t) there is an upper bound $u^{\text{ub}}_{\text{buy}}(B,t)$ on the utility of any adversary who spends at least budget B over exactly t rounds (Lemma 19). From Lemma 16, we obtain a lower bound on the utility $u^{\text{l}}_{\text{honest}}$ of an optimal passive adversary. Since $u^{\text{l}}_{\text{honest}} \geq 0$, by Lemma 20 the maximum in the theorem statement exists.

Then, we eliminate any incentive for adversaries to fork by letting $\text{cutOff} > \ell + \max_{(B,t):u^{\text{ub}}_{\text{buy}}(B,t)>u^{\text{l}}_{\text{honest}}} W(B,t,\ell)$, since no adversary can create a fork of more than ℓ blocks except within a period of $R(B,t,\ell)$ rounds (Lemma 18), without his utility being lower than an passive adversary's. We achieve consistency by using the consistency parameter of $W(B,t,\ell)$ blocks, the number of blocks created within this period of rounds. Liveness (that is, chain growth and chain quality ensured by the ledger) follows from [14]. Now the only parameters affecting our adversary's utility are the block rewards, corruption costs, and mining costs, as we have eliminated forks. Again we work in the $\mathcal{F}_{\text{STX}}$ hybrid which abstracts mining for blocks as an ideal lottery. The cost of playing such a lottery is $\texttt{mcost}$ and the reward for winning the lottery *and* inserting the block into the ledger at round r is $\texttt{breward}(r) \cdot \texttt{CR}$. Importantly, one cannot get this block reward twice, as attacks that cause forks have been eliminated. In this case, all payoff-inducing events made by the adversary in the real world can be matched by an adversary who invokes

an equivalent event in the dummy world with the same payoffs (corruption, mining, and block reward). This implies attack-payoff security.

□

We remark the protocol is not *strong* attack-payoff secure: Recall a front-running adversary always maximally delays honest parties' messages. Intuitively, this reduces the mining power in the system, which delays the time between a block is broadcasted, and when it becomes part of the ledger. In effect, this reduces the payoff for block rewards for utilities with limited horizons.

5.6 Concrete values used in graphs

We discuss the parameters used in the figures. Utility is computed in US dollars (USD). While our results are stated for the Bitcoin protocol, our analyses are in fact quite general, and work for Nakamoto-style PoW blockchains such as Ethereum (Classic) among others. Our graphs use numbers for the popular currency Ethereum Classic, as it has been the victim of several major 51% double-spending attacks whereas Bitcoin was not. For the numbers to be more accurate, we use recent (as of writing, Feb. 2021) data.

Let time be measured in seconds and fiat currency be measured in USD for simplicity. Even basic mining rigs can make millions of hashes per second, thus we model each party/mining rig as capable of 435MH/s (mega hashes per second), the approximate hashing power of a rig with eight AMD RX 5700 XT graphic cards, which costs about $3000 and uses about 1300W. We use a electricity cost of $0.13/kWh (average cost in the US) to compute the cost of mining with this rig as $\texttt{mcost} = \$0.000047$. From the Nicehash marketplace we obtain the cost to rent 435MH (our basic unit of hashing power per party), which is about 1.96 BTC/TH/day (TH is terahash) at the time of writing. Since the price of Bitcoin (BTC) fluctuates wildly, we consider two cases: the "expensive" case, of $50,000 USD/BTC ($\texttt{rcost} = \$0.00045$), and the "cheap" case, of $22,000 USD/BTC ($\texttt{rcost} = \$0.00022$). We consider utility functions with limited horizons; here consider an attacker who is only incentivized to make blocks before some time t. Thus, we set the payoff for making blocks as $\texttt{breward}(x) = \$54.4$ (which is the block reward given price of ETC at $\$17$) if $x < t$ and $\texttt{breward}(x) = 0$ otherwise. We set the payoff for double-spending to $\$1$ million, a relatively modest amount (e.g., the equivalent of 5.6 million dollars in Ethereum Classic had been stolen from an exchange on August 1, 2020).

In our figures we will consider a maximum total hashing power in the system of 18091 mining rigs (each with hashing power 435MH/s), which is the approximate current hashrate of Ethereum Classic. The probability of mining success for one rig per second is $p = 0.00000435$, computed via the current difficulty of the blockchain. We set network delay $\Delta = 100$ms and set window size cutOff $= 500$ blocks using the deposit and withdrawal confirmation time of the OKEx exchange for ETC (100 and 400 blocks respectively).

5.7 Chapter summary and future work

In this chapter we analyze in the rational protocol design (RPD) framework the problem of *51% double-spending attacks* on Nakamoto-style proof-of-work blockchains. We identify an issue with the utility function in the previous work, which we call "unbounded incentives", and alter their utility function to avoid this issue. This allows us to both model protocol/utility parameters where attacking is incentivized, and parameters under which honest behavior is the preferred strategy. The latter, which is a general statement about all rational poly-time attackers, is in particular a demonstration of the power of RPD over other rational analyses. Since our work is in the simpler model of fixed difficulty blockchains, an immediate open question is to expand our analyses to variable difficulty. Lastly, an interesting future direction would be to consider other more complex utility functions, such as those with additional time-dependent cost or reward parameters.

Chapter 6

Conclusion

This thesis has shown how incentive-driven mechanisms can lead to theoretically interesting caveats that can be resolved by analyzing the problem through an alternate framework or set of underlying assumptions.

In Chapter 3, we saw the limitation of *differential privacy* (DP) when encountering deterministic functions. We introduced *exact distributional differential privacy* (eDDP) as a new framework for comparing the privacy of different voting rules. We created a new *trails technique* for analyzing the eDDP of common voting rules, which revealed the encouraging message on the good privacy of most voting rules, and a dichotomy theorem for the class of *generalized scoring rules* (GSR).

In Chapter 4, we showed impossibility results on fairness and identifiable abort that stem from the mediated model in *collusion-preserving* (CP) protocols. Moving away from the mediated model and towards a more decentralized setting, we employed stateful hardware tokens and authenticated broadcast as the basis of our new CP protocol. Matching the fallback security in the mediated model under a corrupt mediator, we altered our CP protocol to provide similar fallback security—and the previously impossible fairness and identifiable abort—when tokens are compromised or when the adversary aborts. Finally, we created a simple penalization scheme that forces misbehaving parties to pay a financial collateral, and proved this indeed deters collusion in (a slightly modified version of) the *rational protocol design* (RPD) framework.

We continued our study of the RPD framework in Chapter 5. We extended the previous work on RPD analysis of the Bitcoin backbone protocol to consider *51% attacks*, where a majority-power adversary breaks the consistency property of the blockchain, creates a fork, and double-spends transactions. We pointed out the issue of *unbounded incentives* in the previous work which led to the unrealistic conclusion that 51% attacks are never profitable, and modified the underlying utility function of their analysis to rectify this issue. This results in both a model that describes scenarios where adversaries are incentivized to attack, and a guide on how to deter attacks by increasing waiting time for transactions.

There are many more fascinating problems waiting to be discovered and addressed in the areas discussed in this thesis. In fact, this thesis serves as a starting step towards the areas of deterministic voting privacy, decentralized collusion-preserving computation, and rational protocol design analysis blockchain incentives. Below, we suggest a few of such next steps.

- Extend the study of voting privacy to general (ϵ, δ), and non-i.i.d. distributions.
- Study the privacy of other high-stakes social choice procedures such as matching and resource allocation.
- Study the privacy of publishing other election data, such as demographic information, and interpret their consequences.
- Adapt our CP protocols and penalization schemes to *cryptographic reverse firewalls* [108].
- Extend the CF/CP model and protocol to incorporate side-channel attacks.
- Expand our 51% attacks analysis to blockchains with variable difficulty, incorporating results from e.g., [44, 67].
- Model additional time-dependent cost or reward parameters in the blockchain utility function, such as incentives relating to transaction fees.

Bibliography

[1] CoinDesk: 51% attacks archives. https://www.coindesk.com/tag/51-attack.

[2] CoinDesk: Ethereum classic's mess solution won't provide 'robust' security against 51% attacks: Report. https://www.coindesk.com/ethereum-classic-mess-security-51-attacks-report.

[3] Forkast: Rented hash power for 51% attacks is a 'huge vulnerability' for proof-of-work blockchains, says etc labs ceo. https://forkast.news/hash-power-51-attack-rent-huge-vulnerability-proof-of-work-blockchain/.

[4] Investopedia: 51% attack. https://www.investopedia.com/terms/1/51-attack.asp.

[5] Poker Fortress: How online poker sites prevent collusion: The complete guide. https://pokerfortress.com/how-online-poker-sites-prevent-collusion/.

[6] United States Census Bureau: Voting and Registration Data Tables. https://www.census.gov/topics/public-sector/voting/data/tables.html.

[7] Pokerology: Cheating & collusion at online poker rooms. http://www.pokerology.com/articles/online-poker-cheating-and-collusion/, 2010.

[8] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 524–540. Springer, Heidelberg, Aug. 2009.

[9] J. Alwen, J. Katz, U. Maurer, and V. Zikas. Collusion-preserving computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 124–143. Springer, Heidelberg, Aug. 2012.

[10] J. Alwen, R. Ostrovsky, H.-S. Zhou, and V. Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 763–780. Springer, Heidelberg, Aug. 2015.

[11] J. Alwen, a. shelat, and I. Visconti. Collusion-free protocols in the mediated model. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 497–514. Springer, Heidelberg, Aug. 2008.

[12] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.

[13] G. Avarikioti, L. Käppeli, Y. Wang, and R. Wattenhofer. Bitcoin security under temporary dishonest majority. In I. Goldberg and T. Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 466–483. Springer, Heidelberg, Feb. 2019.

[14] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Consensus redux: Distributed ledgers in the face of adversarial supremacy. Cryptology ePrint Archive, Report 2020/1021, 2020. `https://eprint.iacr.org/2020/1021`.

[15] C. Badertscher, J. A. Garay, U. Maurer, D. Tschudi, and V. Zikas. But why does it work? A rational protocol design treatment of bitcoin. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 34–65. Springer, Heidelberg, Apr. / May 2018.

[16] C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 913–930. ACM Press, Oct. 2018.

[17] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, Aug. 2017.

[18] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. Cryptology ePrint Archive, Report 2017/149, 2017. `https://eprint.iacr.org/2017/149`.

[19] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi. Secure multiparty computation from SGX. In A. Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 477–497. Springer, Heidelberg, Apr. 2017.

[20] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi. Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 245–260, Los Alamitos, CA, USA, mar 2016. IEEE Computer Society.

[21] R. Bassily, A. Groce, J. Katz, and A. Smith. Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy. In *54th FOCS*, pages 439–448. IEEE Computer Society Press, Oct. 2013.

[22] R. Bassily and A. D. Smith. Local, private, efficient protocols for succinct histograms. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 127–135. ACM Press, June 2015.

[23] F. Benhamouda and H. Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, Apr. / May 2018.

[24] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, Aug. 2014.

[25] I. Bentov, R. Kumaresan, and A. Miller. Instantaneous decentralized poker. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, Dec. 2017.

[26] D. Bernoulli. Exposition of a new theory on the measurement of risk. *Econometrica*, 22(1):23–36, 1954.

[27] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta. Noiseless database privacy. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 215–232. Springer, Heidelberg, Dec. 2011.

[28] E. Birrell and R. Pass. Approximately strategy-proof voting. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, page 67–72. AAAI Press, 2011.

[29] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[30] A. Blum, K. Ligett, and A. Roth. A learning theory approach to non-interactive database privacy. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 609–618. ACM Press, May 2008.

[31] S. Bradshaw and P. N. Howard. Challenging truth and trust: A global inventory of organized social media manipulation. *The Computational Propaganda Project*, 1, 2018.

[32] L. Brünjes, A. Kiayias, E. Koutsoupias, and A. Stouka. Reward sharing schemes for stake pools. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 256–275. IEEE, 2020.

[33] E. Budish. The economic limits of bitcoin and the blockchain. Technical report, National Bureau of Economic Research, 2018.

[34] V. Buterin. A next-generation smart contract and decentralized application platform, 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`.

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

[36] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `https://eprint.iacr.org/2000/067`.

[37] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[38] R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `https://eprint.iacr.org/2003/239`.

[39] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In S. P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, Feb. 2007.

[40] R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, Nov. 2014.

[41] R. Canetti and M. Vald. Universally composable security with local adversaries. In I. Visconti and R. D. Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 281–301. Springer, Heidelberg, Sept. 2012.

[42] I. Caragiannis, A. Procaccia, and N. Shah. Modal ranking: A uniquely robust voting rule. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28, Jun. 2014.

[43] M. Carlsten, H. A. Kalodner, S. M. Weinberg, and A. Narayanan. On the instability of bitcoin without the block reward. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 154–167. ACM Press, Oct. 2016.

[44] T.-H. H. Chan, N. Ephraim, A. Marcedone, A. Morgan, R. Pass, and E. Shi. Blockchain with varying number of players. Cryptology ePrint Archive, Report 2020/677, 2020. `https://eprint.iacr.org/2020/677`.

[45] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[46] K.-M. Chung, T.-H. H. Chan, T. Wen, and E. S. (random author ordering). Game-theoretically fair leader election in o(log log n) rounds under majority coalitions. Cryptology ePrint Archive, Report 2020/1591, 2020. `https://eprint.iacr.org/2020/1591`.

[47] M. Ciampi, Y. Lu, and V. Zikas. Collusion-preserving computation without a mediator. In *Computer Security Foundations Symposium (CSF)*. IEEE, 2022.

[48] J. Clayworth and J. Noble. USA Today: Iowa caucus coin flip count unknown. https://www.usatoday.com/story/news/politics/onpolitics/2016/02/03/iowa-caucus-coin-flip-count-unknown/79748182/, 2016.

[49] R. Cohen, J. A. Garay, and V. Zikas. Broadcast-optimal two-round MPC. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 828–858. Springer, Heidelberg, May 2020.

[50] V. Costan and S. Devadas. Intel SGX explained. Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint.iacr.org/2016/086.

[51] P. Daian, R. Pass, and E. Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In I. Goldberg and T. Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Heidelberg, Feb. 2019.

[52] I. Damgård, J. B. Nielsen, and D. Wichs. Universally composable multiparty computation with partially isolated parties. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 315–331. Springer, Heidelberg, Mar. 2009.

[53] B. David, R. Dowsley, and M. Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In S. Meiklejohn and K. Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 500–519. Springer, Heidelberg, Feb. / Mar. 2018.

[54] B. David, R. Dowsley, and M. Larangeira. ROYALE: A framework for universally composable card games with financial rewards and penalties enforcement. In I. Goldberg and T. Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 282–300. Springer, Heidelberg, Feb. 2019.

[55] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[56] N. Döttling, D. Kraschewski, and J. Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 164–181. Springer, Heidelberg, Mar. 2011.

[57] Y. Duan. Privacy without noise. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, page 1517–1520, New York, NY, USA, 2009. Association for Computing Machinery.

[58] C. Dwork. Differential privacy (invited paper). In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, Heidelberg, July 2006.

[59] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, Mar. 2006.

[60] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, Aug. 2014.

[61] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE Computer Society Press, May 2015.

[62] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014.

[63] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov. IRON: Functional encryption using intel SGX. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 765–782. ACM Press, Oct. / Nov. 2017.

[64] J. A. Garay, J. Katz, R. Kumaresan, and H.-S. Zhou. Adaptively secure broadcast, revisited. In C. Gavoille and P. Fraigniaud, editors, *30th ACM PODC*, pages 179–186. ACM, June 2011.

[65] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th FOCS*, pages 648–657. IEEE Computer Society Press, Oct. 2013.

[66] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, Apr. 2015.

[67] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, Aug. 2017.

[68] J. A. Garay, A. Kiayias, R. M. Ostrovsky, G. Panagiotakos, and V. Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 129–158. Springer, Heidelberg, May 2020.

[69] S. Garfinkel, J. M. Abowd, and C. Martindale. Understanding database reconstruction attacks on public data. *Communications of the ACM*, 62(3):46–53, 2019.

[70] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 3–16. ACM Press, Oct. 2016.

[71] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally utility-maximizing privacy mechanisms. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 351–360. ACM Press, May / June 2009.

[72] O. Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[73] S. Goldwasser and M. Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT*, 1999:1999, 1996.

[74] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56. Springer, Heidelberg, Aug. 2008.

[75] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

[76] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[77] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, Feb. 2010.

[78] W. H. Greene. Econometric analysis. *Econometric analysis*, 2003.

[79] A. D. Groce. *New notions and mechanisms for statistical privacy*. PhD thesis, 2014.

[80] G.-T. Guilbaud. Les théories de l'intérêt général et le problème logique de l'agrégation. *Economie appliquée*, 1952.

[81] R. Hall, L. Wasserman, and A. Rinaldo. Random differential privacy. *Journal of Privacy and Confidentiality*, 4(2), Mar. 2013.

[82] R. Han, Z. Sui, J. Yu, J. Liu, and S. Chen. Fact and fiction: Challenging the honest majority assumption of permissionless blockchains. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 817–831, New York, NY, USA, 2021. Association for Computing Machinery.

[83] M. Hardt and K. Talwar. On the geometry of differential privacy. In L. J. Schulman, editor, *42nd ACM STOC*, pages 705–714. ACM Press, June 2010.

[84] M. Hay, L. Elagina, and G. Miklau. Differentially private rank aggregation. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 669–677. SIAM, 2017.

[85] M. Hirt and V. Zikas. Adaptively secure broadcast. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, Heidelberg, May / June 2010.

[86] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[87] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, Aug. 2014.

[88] J. Jang and H.-N. Lee. Profitable double-spending attacks. *Applied Sciences*, 10(23):8477, 2020.

[89] S. P. Kasiviswanathan and A. Smith. A note on differential privacy: Defining resistance to arbitrary side information. *CoRR abs/0803.3946*, 2008.

[90] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, Heidelberg, May 2007.

[91] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, Mar. 2013.

[92] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, Aug. 2017.

[93] A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.

[94] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 30–41. ACM Press, Nov. 2014.

[95] R. Kumaresan and I. Bentov. Amortizing secure computation with penalties. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *ACM CCS 2016*, pages 418–429. ACM Press, Oct. 2016.

[96] R. Kumaresan, T. Moran, and I. Bentov. How to use bitcoin to play decentralized poker. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 195–206. ACM Press, Oct. 2015.

[97] L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, Nov. 1981.

[98] D. T. Lee. Efficient, private, and eps-strategyproof elicitation of tournament voting rules. In *IJCAI*, pages 2026–2032, 2015.

[99] M. Lepinski, S. Micali, C. Peikert, and a. shelat. Completely fair sfe and coalition-safe cheap talk. In S. Chaudhuri and S. Kutten, editors, *23rd ACM PODC*, pages 1–10. ACM, July 2004.

[100] M. Lepinski, S. Micali, and a. shelat. Collusion-free protocols. In H. N. Gabow and R. Fagin, editors, *37th ACM STOC*, pages 543–552. ACM Press, May 2005.

[101] S. Leung and E. Lui. Bayesian mechanism design with efficiency, privacy, and approximate truthfulness. In *International Workshop on Internet and Network Economics*, pages 58–71. Springer, 2012.

[102] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, Apr. 2009.

[103] A. Liu, Y. Lu, L. Xia, and V. Zikas. How private are commonly-used voting rules? In J. Peters and D. Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 629–638. PMLR, 03–06 Aug 2020.

[104] Y. Lu, V. Zikas, and C. Badertscher. A rational protocol treatment of 51% attacks. In *CRYPTO*. Springer-Verlag, 2021.

[105] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena. Demystifying incentives in the consensus computer. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 706–719. ACM Press, Oct. 2015.

[106] N. Mattei and T. Walsh. Preflib: A library for preferences. In *International Conference on Algorithmic DecisionTheory*, pages 259–270. Springer, 2013.

[107] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *48th FOCS*, pages 94–103. IEEE Computer Society Press, Oct. 2007.

[108] I. Mironov and N. Stephens-Davidowitz. Cryptographic reverse firewalls. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Heidelberg, Apr. 2015.

[109] T. Moran and G. Segev. David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 527–544. Springer, Heidelberg, Apr. 2008.

[110] E. Mossel, A. D. Procaccia, and M. Z. Rácz. A smooth transition from powerlessness to absolute power. *Journal of Artificial Intelligence Research*, 48:923–951, 2013.

[111] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.

[112] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *S&P*, 2016.

[113] J. B. Nielsen. *On protocol security in the cryptographic model*. Citeseer, 2003.

[114] T. Nilges. *The cryptographic strength of tamper-proof hardware*. PhD thesis, Karlsruhe Institute of Technology, 2015.

[115] R. Pass, L. Seeman, and a. shelat. Analysis of the blockchain protocol in asynchronous networks. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 643–673. Springer, Heidelberg, Apr. / May 2017.

[116] R. Pass and E. Shi. FruitChains: A fair blockchain. In E. M. Schiller and A. A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017.

[117] A. Patra and D. Ravi. On the exact round complexity of secure three-party computation. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 425–458. Springer, Heidelberg, Aug. 2018.

[118] M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *CoRR*, 2011.

[119] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. In J. Grossklags and B. Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 515–532. Springer, Heidelberg, Feb. 2016.

[120] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. In J. Grossklags and B. Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 477–498. Springer, Heidelberg, Feb. 2016.

[121] S. Shang, T. Wang, P. Cuff, and S. Kulkarni. The application of differential privacy for rank aggregation: Privacy and accuracy. In *17th International Conference on Information Fusion (FUSION)*, pages 1–7. IEEE, 2014.

[122] J. Teutsch, S. Jain, and P. Saxena. When cryptocurrencies mine their own business. In J. Grossklags and B. Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 499–514. Springer, Heidelberg, Feb. 2016.

[123] J. M. Varah. A lower bound for the smallest singular value of a matrix. *Linear Algebra and its applications*, 11(1):3–5, 1975.

[124] J. Verini. Vanity Fair: Big brother inc. `https://www.vanityfair.com/news/2007/12/aristotle200712`, 2007.

[125] J. Wang, S. Sikdar, T. Shepherd, Z. Zhao, C. Jiang, and L. Xia. Practical algorithms for multi-stage voting rules with parallel universes tiebreaking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2189–2196, 2019.

[126] L. Xia. Generalized decision scoring rules: Statistical, computational, and axiomatic properties. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, page 661–678, New York, NY, USA, 2015. Association for Computing Machinery.

[127] L. Xia and V. Conitzer. Generalized scoring rules and the frequency of coalitional manipulability. *Electronic Commerce*, 2008.

[128] L. Xia and V. Conitzer. Finite local consistency characterizes generalized scoring rules. In *IJCAI*, pages 336–341, 2009.

Appendix A

Additional Background on the Ledger Functionality (Section 2.5)

A.1 Clock

In Fig. A.1 we model the global clock, which is used to keep track of the current time/round.

The functionality $\mathcal{G}_{\text{CLOCK}}$ is available to all participants. The functionality is parametrized with variable t, a set of parties $\mathcal{P}'$, and a set F of functionalities. For each party $p \in \mathcal{P}'$ it manages variable d_p. For each $\mathcal{F} \in F$ it manages variable $d_{\mathcal{F}}$

Initially, $t := 0$, $\mathcal{P}' := \emptyset$ and $F := \emptyset$.

Synchronization:

- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some party $p \in \mathcal{P}'$ set $d_p := 1$; execute *Round-Update* and forward $(\text{CLOCK-UPDATE}, \text{sid}_C, p)$ to $\mathcal{A}$.
- Upon receiving $(\text{CLOCK-UPDATE}, \text{sid}_C)$ from some functionality $\mathcal{F} \in F$ set $d_{\mathcal{F}} := 1$, execute *Round-Update* and return $(\text{CLOCK-UPDATE}, \text{sid}_C, \mathcal{F})$ to $\mathcal{F}$.
- Upon receiving $(\text{CLOCK-READ}, \text{sid}_C)$ from any participant (including the environment, the adversary, or any ideal—shared or local—functionality) return $(\text{CLOCK-READ}, \text{sid}_C, t)$ to the requestor.

Procedure Round-Update:

If $d_{\mathcal{F}} := 1$ for all $\mathcal{F} \in F$ and $d_p = 1$ for all honest p in $\mathcal{P}'$, then set $t := t + 1$ and reset $d_{\mathcal{F}} := 0$ and $d_p := 0$ for all parties in $\mathcal{P}'$.

Figure A.1: The functionality $\mathcal{G}_{\text{CLOCK}}$

A.2 Multicast

In Fig. A.2 we describe the multicast functionality with network delay Δ.

A.3 Ledger functionality

In Fig. A.3, we describe the ledger functionality $\mathcal{G}_{\text{LEDGER}}$ from [17].

A.4 Weak ledger functionality

In Fig. A.5, we formally describe this weaker Bitcoin ledger functionality $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ from [15].

The functionality $\mathcal{F}_{\text{N-MC}}^{\Delta}$ is parametrized with a set possible senders and receivers $\mathcal{P}$. Any newly registered (resp. deregistered) party is added to (resp. deleted from) $\mathcal{P}$.

- *Honest sender multicast:*
  Upon receiving a message $(\text{MULTICAST}, \mathsf{sid}, m)$ from some $p_s \in \mathcal{P}$, where $\mathcal{P} = \{p_1, \ldots, p_n\}$ denotes the current party set, choose n new unique message-IDs $\mathsf{mid}_1, \ldots, \mathsf{mid}_n$, initialize $2n$ new variables $D_{\mathsf{mid}_1} := D_{\mathsf{mid}_1}^{MAX} \ldots := D_{\mathsf{mid}_n} := D_{\mathsf{mid}_n}^{MAX} := 1$, set $\vec{M} := \vec{M}||(m, \mathsf{mid}_1, D_{\mathsf{mid}_1}, p_1)|| \ldots ||(m, \mathsf{mid}_n, D_{\mathsf{mid}_n}, p_n)$, and send $(\text{MULTICAST}, \mathsf{sid}, m, p_s, (p_1, \mathsf{mid}_1), \ldots, (p_n, \mathsf{mid}_n))$ to the adversary.

- *Adversarial sender (partial) multicast:*
  Upon receiving a message $(\text{MULTICAST}, \mathsf{sid}, (m_{i_1}, p_{i_1}), \ldots, (m_{i_\ell}, p_{i_\ell})$ from the adversary with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{P}$, choose ℓ new unique message-IDs $\mathsf{mid}_{i_1}, \ldots, \mathsf{mid}_{i_\ell}$, initialize ℓ new variables $D_{\mathsf{mid}_{i_1}} := D_{\mathsf{mid}_{i_1}}^{MAX} := \ldots := D_{\mathsf{mid}_{i_\ell}} := D_{\mathsf{mid}_{i_\ell}}^{MAX} := 1$, set $\vec{M} := \vec{M}||(m_{i_1}, \mathsf{mid}_{i_1}, D_{\mathsf{mid}_{i_1}}, p_{i_1})|| \ldots ||(m_{i_\ell}, \mathsf{mid}_{i_\ell}, D_{\mathsf{mid}_{i_\ell}}, p_\ell)$, and send $((\text{MULTICAST}, \mathsf{sid}, (m_{i_1}, p_{i_1}, \mathsf{mid}_{i_1}), \ldots, (m_{i_\ell}, p_{i_\ell}, \mathsf{mid}_{i_\ell})$ to the adversary.

- *Honest party fetching:*
  Upon receiving a message $(\text{FETCH}, \mathsf{sid})$ from $p_i \in \mathcal{P}$ (or from $\mathcal{A}$ on behalf of p_i if p_i is corrupted):

  1. For all tuples $(m, \mathsf{mid}, D_{\mathsf{mid}}, p_i) \in \vec{M}$, set $D_{\mathsf{mid}} := D_{\mathsf{mid}} - 1$.
  2. Let $\vec{M}_0^{p_i}$ denote the subvector $\vec{M}$ including all tuples of the form $(m, \mathsf{mid}, D_{\mathsf{mid}}, p_i)$ with $D_{\mathsf{mid}} = 0$ (in the same order as they appear in $\vec{M}$). Delete all entries in $\vec{M}_0^{p_i}$ from $\vec{M}$, and send $\vec{M}_0^{p_i}$ to p_i.

- *Adding adversarial delays:*
  Upon receiving a message $(\text{DELAYS}, \mathsf{sid}, (T_{\mathsf{mid}_{i_1}}, \mathsf{mid}_{i_1}), \ldots, (T_{\mathsf{mid}_{i_\ell}}, \mathsf{mid}_{i_\ell}))$ do the following for each pair $(T_{\mathsf{mid}_{i_j}}, \mathsf{mid}_{i_j})$ in this message:
  If $D_{\mathsf{mid}_{i_j}}^{MAX} + T_{\mathsf{mid}_{i_j}} \leq \Delta$ and mid is a message-ID registered in the current $\vec{M}$, set $D_{\mathsf{mid}_{i_j}} := D_{\mathsf{mid}_{i_j}} + T_{\mathsf{mid}_{i_j}}$ and set $D_{\mathsf{mid}_{i_j}}^{MAX} := D_{\mathsf{mid}_{i_j}}^{MAX} + T_{\mathsf{mid}_{i_j}}$; otherwise, ignore this pair.

- *Adversarially reordering messages:*
  Upon receiving a message $(\text{SWAP}, \mathsf{sid}, \mathsf{mid}, \mathsf{mid}')$ from the adversary, if mid and mid′ are message-IDs registered in the current $\vec{M}$, then swap the triples $(m, \mathsf{mid}, D_{\mathsf{mid}}, \cdot)$ and $(m, \mathsf{mid}', D_{\mathsf{mid}'}, \cdot)$ in $\vec{M}$. Return $(\text{SWAP}, \mathsf{sid})$ to the adversary.

Figure A.2: The functionality $\mathcal{F}_{\text{N-MC}}^{\Delta}$

The functionality manages variables state, $\texttt{NxtBC}$, buffer, τ_L, and $\vec{\tau}_{\texttt{state}}$, as described above. The variables are initialized as follows: state $:= \vec{\tau}_{\texttt{state}} := \texttt{NxtBC} := \varepsilon$, buffer $:= \emptyset$, $\tau_L = 1$. The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$. The set $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered, it is added to all $\mathcal{P}_{DS}$ (hence also to $\mathcal{H}$ and $\mathcal{P}$) and the current time of registration is also recorded; similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ and $\mathcal{P}_{DS}$.

For each party $p_i \in \mathcal{P}$ the functionality maintains a pointer $\texttt{pt}_i$ (initially set to 1) and a current-state view $\text{state}_i := \varepsilon$ (initially set to empty). The functionality also keeps track of the timed honest-input sequence in a vector $\vec{\mathcal{I}}_H^T$ (initially $\vec{\mathcal{I}}_H^T := \varepsilon$).

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, t)$ set $\tau_L := t$ and do the following:

1. If I was received by an honest party $p_i \in \mathcal{P}$:
   (a) Set $\vec{\mathcal{I}}_H^T := \vec{\mathcal{I}}_H^T || (I, p_i, \tau_L)$;
   (b) Compute $\vec{N} = (\vec{N}_1, \ldots, \vec{N}_\ell) := \texttt{ExtendPolicy}(\vec{\mathcal{I}}_H^T, \text{state}, \texttt{NxtBC}, \text{buffer}, \vec{\tau}_{\texttt{state}})$ and if $\vec{N} \neq \varepsilon$ set state $:=$ state$||\text{Blockify}(\vec{N}_1)|| \ldots ||\text{Blockify}(\vec{N}_\ell)$ and $\vec{\tau}_{\texttt{state}} := \vec{\tau}_{\texttt{state}} || \tau_L^\ell$, where $\tau_L^\ell = \tau_L || \ldots, || \tau_L$.
   (c) $\forall$ $\texttt{BTX} \in$ buffer: if $\text{Validate}(\texttt{BTX}, \text{state}, \text{buffer}) = 0$ then delete $\texttt{BTX}$ from buffer.
   (d) If there exists $j \in [\ell]$ with $p_{i_j} \in \mathcal{H} \setminus \mathcal{P}_{DS}$: $|\text{state}| - \hat{\texttt{pt}}_{i} \leq \texttt{windowSize}$ or $\hat{\texttt{pt}}_{i_j} \geq |\text{state}_{i_j}|$, then for every $j \in [\ell]$ set $\texttt{pt}_i := |\text{state}| - |\vec{\mathbf{N}}|$ for every $p_i \in \mathcal{P}$.
2. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that were registered at time $\tau' \leq \tau_L - \texttt{Delay}$. Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \hat{\mathcal{P}}$.
3. Depending on the above input I and its sender's ID, $\mathcal{G}_{\text{LEDGER}}$ executes the corresponding code from the following list:
   – *Submiting a transaction:* If $I = (\text{SUBMIT}, \text{sid}, \texttt{x})$ and is received from a party $p_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party p_i) do the following
      (a) Choose a unique transaction ID $\texttt{txid}$ and set $\texttt{BTX} := (x, \texttt{txid}, \tau_L, p_i)$
      (b) If $\text{Validate}(\texttt{BTX}, \text{state}, \text{buffer}) = 1$, then buffer $:=$ buffer $\cup \{\texttt{BTX}\}$.
      (c) Send $(\text{SUBMIT}, \texttt{BTX})$ to $\mathcal{A}$.

Figure A.3: The functionality $\mathcal{G}_{\text{LEDGER}}$ (Part 1)

(Continued from description of $\mathcal{G}_{\text{LEDGER}}$, Part 1)

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, t)$ set $\tau_L := t$ and do the following:

3. Depending on the above input I and its sender's ID, $\mathcal{G}_{\text{LEDGER}}$ executes the corresponding code from the following list:

   - *Reading the state:* If $I = (\text{READ}, \text{sid})$ is received from a party $p_i \in \mathcal{P}$ then set $\text{state}_i := \text{state}|_{\min\{\texttt{pt}_i, |\text{state}|\}}$ and return $(\text{READ}, \text{sid}, \text{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\text{state}, \text{buffer}, \vec{\mathcal{I}}_H^T)$ to $\mathcal{A}$.
   - *Updating the state:* If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $p_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}_H^T$ as above) $\texttt{predict-time}(\vec{\mathcal{I}}_H^T) = \hat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.
   - *The adversary proposing the next block:*
     If $I = (\text{NEXT-BLOCK}, (\texttt{txid}_1, \ldots, \texttt{txid}_\ell))$ is sent from the adversary, update $\texttt{NxtBC}$ as follows:
     (a) Set $\texttt{NxtBC} := \varepsilon$.
     (b) For $i = 1, \ldots, \ell$ do: if there exists $\texttt{BTX} := (x, \texttt{txid}, \text{minerID}, \tau_L, p_i) \in$ buffer with ID $\texttt{txid} = \texttt{txid}_i$ then set $\texttt{NxtBC} := \texttt{NxtBC}||\texttt{txid}_i$.
     (c) Output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$.
   - *The adversary setting state-slackness:*
     If $I = (\text{SET-SLACK}, (p_{i_1}, \hat{\texttt{pt}}_{i_1}), \ldots, (p_{i_\ell}, \hat{\texttt{pt}}_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
     (a) If for all $j \in [\ell]$: $|\text{state}| - \hat{\texttt{pt}}_{i_j} \leq \texttt{windowSize}$ and $\hat{\texttt{pt}}_{i_j} \geq |\text{state}_{i_j}|$, set $\texttt{pt}_{i_1} := \hat{\texttt{pt}}_{i_1}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.
     (b) Otherwise set $\texttt{pt}_{i_j} := |\text{state}|$ for all $j \in [\ell]$.
   - *The adversary setting the state for desychronized parties:*
     If $I = (\text{DESYNC-STATE}, (p_{i_1}, \text{state}'_{i_1}), \ldots, (p_{i_\ell}, \text{state}'_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\text{state}_{i_j} := \text{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.

Figure A.4: The functionality $\mathcal{G}_{\text{LEDGER}}$ (Part 2)

The functionality $\mathcal{G}^{\tilde{\mathbb{B}}}_{\text{WEAK-LEDGER}}$ manages variables state-tree, NxtBC, buffer, and τ_L, where state-tree is a tree of state blocks. The variables are initialized as follows: state-tree $=$ gen, NxtBC $:= \varepsilon$, buffer $:= \emptyset$, $\tau_L = 0$. For each party $p_i \in \mathcal{P}$ the functionality maintains a pointer $\mathtt{pt}_i$ (initially set to the root of state-tree) which defines the current-state view state_i of p_i. The functionality also keeps track of the timed honest-input sequence in a vector $\vec{\mathcal{I}}^T_H$ (initially $\vec{\mathcal{I}}^T_H := \varepsilon$).

Party Management: The functionality maintains the set of registered parties $\mathcal{P}$, the (sub-)set of honest parties $\mathcal{H} \subseteq \mathcal{P}$, and the (sub-set) of de-synchronized honest parties $\mathcal{P}_{DS} \subset \mathcal{H}$ (following the definition of de-synchronized of [17]). The sets $\mathcal{P}, \mathcal{H}, \mathcal{P}_{DS}$ are all initially set to $\emptyset$. When a new honest party is registered, if it is registered with the clock then it is added to the party sets $\mathcal{H}$ and $\mathcal{P}$ and the current time of registration is also recorded; if the current time is $\tau_L > 0$, it is also added to $\mathcal{P}_{DS}$. Similarly, when a party is deregistered, it is removed from both $\mathcal{P}$ (and therefore also from $\mathcal{P}_{DS}$ or $\mathcal{H}$). The ledger maintains the invariant that it is registered (as a functionality) to the clock whenever $\mathcal{H} \neq \emptyset$. A party is considered fully registered if it is registered with the ledger and the clock.

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \mathsf{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \mathsf{sid}_C, t)$ set $\tau_L := t$ and do the following:

1. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{DS}$ denote the set of desynchronized honest parties that have been registered (continuously) since time $\tau' < \tau_L - \mathtt{Delay}$ (to both ledger and clock). Set $\mathcal{P}_{DS} := \mathcal{P}_{DS} \setminus \hat{\mathcal{P}}$. On the other hand, for any synchronized party $p \in \mathcal{H} \setminus \mathcal{P}_{DS}$, if p is not registered to the clock, then $\mathcal{P}_{DS} \cup \{p\}$.
2. If I was received from an honest party $p_i \in \mathcal{P}$:
   (a) Set $\vec{\mathcal{I}}^T_H := \vec{\mathcal{I}}^T_H || (I, p_i, \tau_L)$;
   (b) Evaluate $\vec{R} = ((\vec{N}_{\mathtt{pt}_1}, \mathtt{pt}_1, \ldots, (\vec{N}_{\mathtt{pt}_k}, \mathtt{pt}_k)) := \mathsf{weakExtendPolicy}(\vec{\mathcal{I}}^T_H, \mathtt{state\text{-}tree}, \mathtt{NxtBC}, \mathsf{buffer})$
   (c) For each pointer $\mathtt{pt}_i$ such that $\vec{N}_{\mathtt{pt}_i} \neq \varepsilon$ add path $\mathsf{Blockify}(\vec{N}_{\mathtt{pt}_i,1}), \ldots, \mathsf{Blockify}(\vec{N}_{\mathtt{pt}_i,\ell})$ to state-tree starting at node $\mathtt{pt}_i$.
   (d) Reset NxtBC $:= \varepsilon$.

Figure A.5: The weak Bitcoin ledger functionality (Part 1)

(Continued from description of $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$, Part 1)

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, t)$ set $\tau_L := t$ and do the following:

3. Depending on the above input I and its sender's ID, $\mathcal{G}^{\mathbb{B}}_{\text{WEAK-LEDGER}}$ executes the corresponding code from the following list:

   - *Submiting a transaction:*
     If $I = (\text{SUBMIT}, \text{sid}, \texttt{tx})$ and is received from a party $p_i \in \mathcal{P}$ or from $\mathcal{A}$ (on behalf of a corrupted party p_i) do the following
     (a) Choose a unique transaction ID `txid` and set $\texttt{BTX} := (\texttt{tx}, \texttt{txid}, \tau_L, p_i)$
     (b) Set buffer := buffer $\cup \{\texttt{BTX}\}$ and send $(\text{SUBMIT}, \texttt{BTX})$ to $\mathcal{A}$.
   - *Reading the state:*
     If $I = (\text{READ}, \text{sid})$ is received from a fully registered party $p_i \in \mathcal{P}$ return $(\text{READ}, \text{sid}, \text{state}_i)$ to the requestor. If the requestor is $\mathcal{A}$ then send $(\texttt{state-tree}, \text{buffer}, \vec{\mathcal{I}}^T_H)$ to $\mathcal{A}$.
   - *Maintaining the ledger state:*
     If $I = (\text{MAINTAIN-LEDGER}, \text{sid}, \text{minerID})$ is received by an honest party $p_i \in \mathcal{P}$ and (after updating $\vec{\mathcal{I}}^T_H$ as above) $\texttt{predict-time}(\vec{\mathcal{I}}^T_H) = \hat{\tau} > \tau_L$ then send $(\text{CLOCK-UPDATE}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$. Else send I to $\mathcal{A}$.
   - *The adversary proposing the next block:*
     If $I = (\text{NEXT-BLOCK}, \texttt{pt}, \text{hFlag}, (\texttt{txid}_1, \ldots, \texttt{txid}_\ell))$ is sent from the adversary, update `NxtBC` as follows:
     (a) Check that `pt` points to a leaf of `state-tree` and set $\text{listOfTxid} \leftarrow \varepsilon$ (otherwise, ignore command)
     (b) For $i = 1, \ldots, \ell$ do: if there exists $\texttt{BTX} := (x, \texttt{txid}, \text{minerID}, \tau_L, p_i) \in$ buffer with ID $\texttt{txid} = \texttt{txid}_i$ then set $\text{listOfTxid} := \text{listOfTxid}||\texttt{txid}_i$.
     (c) Finally, set $\texttt{NxtBC}[\texttt{pt}] := \texttt{NxtBC}[\texttt{pt}]||(\text{hFlag}, \text{listOfTxid})$ and output $(\text{NEXT-BLOCK}, ok)$ to $\mathcal{A}$.

Figure A.6: The weak Bitcoin ledger functionality (Part 2)

(Continued from description of $\mathcal{G}^{\Bbbk}_{\text{WEAK-LEDGER}}$, Part 2)

Upon receiving any input I from any party or from the adversary, send $(\text{CLOCK-READ}, \text{sid}_C)$ to $\mathcal{G}_{\text{CLOCK}}$ and upon receiving response $(\text{CLOCK-READ}, \text{sid}_C, t)$ set $\tau_L := t$ and do the following:

3. Depending on the above input I and its sender's ID, $\mathcal{G}^{\Bbbk}_{\text{WEAK-LEDGER}}$ executes the corresponding code from the following list:

   - *The adversary proposing a fork:*
     If $I = (\text{FORK}, \mathtt{pt}, (\mathtt{txid}_1, \ldots, \mathtt{txid}_\ell))$ is sent from the adversary, update $\mathtt{NxtBC}$ as follows:
     (a) Set $\text{listOfTxid} \leftarrow \varepsilon$
     (b) For $i = 1, \ldots, \ell$ do: if there exists $\mathtt{BTX} := (x, \mathtt{txid}, \text{minerID}, \tau_L, p_i) \in$ buffer with ID $\mathtt{txid} = \mathtt{txid}_i$ then set $\text{listOfTxid} := \text{listOfTxid}||\mathtt{txid}_i$.
     (c) Finally, set $\mathtt{NxtBC}[\mathtt{pt}] := \mathtt{NxtBC}[\mathtt{pt}]||(0, \text{listOfTxid})$ and output (FORK, ok) to $\mathcal{A}$.
   - *The adversary setting state-slackness:*
     If $I = (\text{SET-POINTER}, (p_{i_1}, \hat{\mathtt{pt}}_{i_1}), \ldots, (p_{i_\ell}, \hat{\mathtt{pt}}_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{H} \setminus \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$ do the following:
     (a) If for all $j \in [\ell]$: $\hat{\mathtt{pt}}_{i_j}$ has greater distance than $\mathtt{pt}_{i_j}$ from the root $\mathtt{state\text{-}tree}$, set$\mathtt{pt}_{i_j} := \hat{\mathtt{pt}}_{i_j}$ for every $j \in [\ell]$ and return $(\text{SET-SLACK}, ok)$ to $\mathcal{A}$.
     (b) Otherwise set $\mathtt{pt}_{i_j}$ to the leaf with greatest distance from the root of $\mathtt{state\text{-}tree}$.
   - *The adversary setting the state for desychronized parties:*
     If $I = (\text{DESYNC-STATE}, (p_{i_1}, \text{state}'_{i_1}), \ldots, (p_{i_\ell}, \text{state}'_{i_\ell}))$, with $\{p_{i_1}, \ldots, p_{i_\ell}\} \subseteq \mathcal{P}_{DS}$ is received from the adversary $\mathcal{A}$, set $\text{state}_{i_j} := \text{state}'_{i_j}$ for each $j \in [\ell]$ and return $(\text{DESYNC-STATE}, ok)$ to $\mathcal{A}$.

Figure A.7: The weak Bitcoin ledger functionality (Part 3)