

# How to Use Bitcoin to Design Fair Protocols

Iddo Bentov\*

iddo@cs.technion.ac.il

Ranjit Kumaresan†

ranjit@cs.technion.ac.il

## Abstract

We study a model of fairness in secure computation in which an adversarial party that aborts on receiving output is forced to pay a mutually predefined monetary penalty. We then show how the Bitcoin network can be used to achieve the above notion of fairness in the two-party as well as the multiparty setting (with a dishonest majority). In particular, we propose new ideal functionalities and protocols for fair secure computation and fair lottery in this model.

One of our main contributions is the definition of an ideal primitive, which we call  $\mathcal{F}_{\text{CR}}^*$  (CR stands for “claim-or-refund”), that formalizes and abstracts the exact properties we require from the Bitcoin network to achieve our goals. Naturally, this abstraction allows us to design fair protocols in a hybrid model in which parties have access to the  $\mathcal{F}_{\text{CR}}^*$  functionality, and is otherwise independent of the Bitcoin ecosystem. We also show an efficient realization of  $\mathcal{F}_{\text{CR}}^*$  that requires only two Bitcoin transactions to be made on the network.

Our constructions also enjoy high efficiency. In a multiparty setting, our protocols only require a constant number of calls to  $\mathcal{F}_{\text{CR}}^*$  per party on top of a standard multiparty secure computation protocol. Our fair multiparty lottery protocol improves over previous solutions which required a quadratic number of Bitcoin transactions.

**Keywords:** Fair exchange, Secure computation, Bitcoin.

## 1 Introduction

**Secure computation** enables a set of mutually distrusting parties to carry out a distributed computation without compromising on privacy of inputs or correctness of the end result. Indeed, secure computation is widely applicable to variety of everyday tasks ranging from electronic auctions to privacy-preserving data mining. Showing feasibility [95, 49, 19, 32] of this seemingly impossible-to-achieve notion has been one of the most striking contributions of modern cryptography. However, definitions of secure computation [48] do vary across models, in part owing to general impossibility results for fair coin-tossing [35]. In settings where the majority of the participating parties are dishonest (including the two party setting), a protocol for secure computation protocols is not required to guarantee important properties such as guaranteed output delivery or fairness.<sup>1</sup> Addressing this deficiency is critical if secure computation is to be widely adopted in practice, especially given the current interest in practical secure computation. Needless to say, it is not very appealing for an honest party to invest time and money to carry out a secure computation protocol until the very end, only to find out that its adversarial partner has aborted the protocol after learning the output.

---

\*Supported by funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement number 240258.

†Supported by funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement number 259426.

<sup>1</sup>Fairness guarantees that if one party receives output then all parties receive output. Guaranteed output delivery ensures that an adversary cannot prevent the honest parties from computing the function.

**Fair exchange** of digital commodities is a well-motivated and well-studied problem. Loosely speaking, in the problem of fair exchange, there are two (or more) parties that wish to exchange digital commodities (e.g., signed contracts) in a fair manner, i.e., either both parties complete the exchange, or none do. A moment’s thought reveals that fair exchange is indeed a special subcase of fair secure computation. Unfortunately, as is the case with fair secure computation, it is known that fair exchange in the standard model cannot be achieved [22, 35]. However, solutions for fair exchange were investigated and proposed in a variety of weaker models, most notably in the optimistic model mentioned below. Typically such solutions require cryptosystems with some tailor-made properties, and employ tools of generic secure computation only sparingly (see [23, 68]) in part owing to the assumed inefficiency of secure computation protocols. Recent years, however, have witnessed a tremendous momentum shift in practical secure computation (see [87, 67, 74] and references therein). Given the zeitgeist, it may seem that solving the problem of fair exchange as a subcase of fair secure computation is perhaps the right approach to take.<sup>2</sup> Unfortunately as described earlier, fair secure computation is impossible.

**Workarounds.** Indeed, several workarounds have been proposed in the literature to counter adversaries that may decide to abort possibly depending on the outcome of the protocol. The most prominent lines of work include gradual release mechanisms, optimistic models, and partially fair secure computation. Gradual release mechanisms ensure that at any stage of the protocol, the adversary has not learned much more about the output than honest parties. Optimistic models allow parties to pay a subscription fee to a trusted server that can be contacted to restore fairness whenever fairness is breached. Partially fair secure computation provides a solution for secure computation where fairness may be breached but only with some parameterizable (inverse polynomial) probability. In all of the above solutions, one of two things hold: either (1) parties have to run a secure computation protocol that could potentially be much more expensive (especially in the number of rounds) than a standard secure computation protocol, or (2) an external party must be trusted to not collude with the adversary. Further, when an adversary aborts, the honest parties have to expend *extra effort* to restore fairness, e.g., the trusted server in the optimistic model needs to be contacted each time fairness is breached. In summary, in all these works, (1) the honest party has to expend extra effort, and (2) the adversary essentially gets away with cheating.<sup>3</sup>

Ideally, rather than asking an honest party to invest additional time and money whenever fairness is (expected to be) breached by the adversary, one would expect “fair” mechanisms to compensate an honest party in such situations. Indeed, this point-of-view was taken by several works [73, 69, 16]. These works ensure that an honest party would be monetarily compensated whenever a dishonest party aborts. In practice, such mechanisms would be effective if the compensation amount is rightly defined. Note that in contrast to the optimistic model, here the honest party is not guaranteed to get output, but still these works provide a reasonable and practical notion of fairness. Perhaps the main drawback of such works is their dependance on e-cash systems (which unfortunately are not widely adopted yet) or central bank systems which need to be completely trusted.

**Bitcoin** [84] is a peer-to-peer network that uses the power of cryptography to emulate a trusted bank (among other things). Its claim to fame is that it is the first practical decentralized digital currency system (which also provides some level of anonymity for its users). A wide variety of electronic transactions take place on the Bitcoin network. As an illustrative example, consider the case of (multiparty) lotteries which are typically conducted by gambling websites (e.g., SatoshiDice). Note that such a lottery requires the participants to trust the gambling website to properly conduct the lottery which may be unreasonable in some cases (and further necessitates paying a house edge). One might wonder if secure computation would provide a natural solution for multiparty lotteries over Bitcoin. Unfortunately, our understanding of Bitcoin is diminished by a lack of abstraction of what the Bitcoin network provides. Consequently there exist relatively very few works that provide any *constructive* uses of Bitcoin [34, 4, 11].

---

<sup>2</sup>A similar parallel may be drawn to the practicality of secure computation itself. Special purpose protocols for secure computation were exclusively in vogue until very recently. However, a number of recent works have shown that generic secure computation can be much more practical [77, 61].

<sup>3</sup>This is especially true in today’s world where cheap digital pseudonyms [41] are available.

**Our contributions.** Conceptually, our work provides the *missing piece* that simultaneously allows (1) designing protocols of fair secure computation that rely on Bitcoin (and not a trusted central bank), and (2) designing protocols for fair lottery on Bitcoin that use secure computation (and not a trusted gambling website). Our model of fairness is essentially the same as in [4, 73, 69, 3] in that we wish to monetarily penalize an adversary that aborts the protocol after learning the output. We distinguish ourselves from most prior work by providing a *formal treatment*, namely specifying formal security models and definitions, and *proving* security of our constructions. In addition, we extensively consider the *multiparty* setting, and construct protocols that are both more efficient as well as provably secure (in our new model). Our clear abstraction of the functionality that we require from Bitcoin network enables us to not only design modular protocols, but also allow easy adaptations of our solutions to settings other than the Bitcoin network (e.g., Zerocash, Litecoin, or even PayPal or a central trusted bank).<sup>4</sup> Our main contributions include providing formal definitions and efficient realizations for:

- **Claim-or-refund functionality  $\mathcal{F}_{\text{CR}}^*$ .** A simple yet powerful two-party primitive that accepts deposits from a “sender” and conditionally transfers the deposit to a “receiver.” If the receiver defaults, then the deposit is returned to the sender after a prespecified time. We provide a Bitcoin protocol for realizing this functionality that requires parties to make only two transactions on the Bitcoin network.
- **Secure computation with penalties  $\mathcal{F}_f^*$ .** In a  $n$ -party setting, a protocol for secure computation with penalties guarantees that if an adversary aborts after learning the output but before delivering output to honest parties, then *each* honest party is compensated by a prespecified amount. We show how to construct such a protocol in the  $(\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{CR}}^*)$ -hybrid model that requires only  $O(n)$  rounds<sup>5</sup> and  $O(n)$  calls to  $\mathcal{F}_{\text{CR}}^*$ .
- **Secure lottery with penalties  $\mathcal{F}_{\text{lot}}^*$ .** In a multiparty setting, a protocol for secure lottery with penalties guarantees that if an adversary aborts after learning the outcome of the lottery but before revealing the outcome to honest parties, then *each* honest party is compensated by a prespecified amount equal to the lottery prize. We show how to construct such a protocol in the  $(\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{CR}}^*)$ -hybrid model that requires only  $O(n)$  rounds and  $O(n)$  calls to  $\mathcal{F}_{\text{CR}}^*$ .

**Potential impact.** We hope that our work will encourage researchers to undertake similar attempts at formalizing other important properties of the Bitcoin network, and perhaps even develop a fully rigorous framework for secure computations that involve financial transactions. Also, we design our protocols in a hybrid model, thus enabling us to take advantage of advances in practical secure computation. One reason to do this was because we are (perhaps unreasonably) optimistic that our protocols will have a practical impact on the way electronic transactions are conducted over the internet and the Bitcoin network.

**Related work.** Most related to our work are the works of Back and Bentov [11] and Andrychowicz *et al.* [4, 3]. Indeed, our work is heavily inspired by [11, 4] who, to the best of our knowledge, were the first to propose fair two-party (resp. multiparty) lottery protocols over the Bitcoin network. We point out that the  $n$ -party lottery protocols of [4] require quadratic number of transactions to be made on the Bitcoin network. In contrast our protocols require only a linear number of Bitcoin transactions. In a followup work [3] that is concurrent to and independent of ours, the authors of [4] propose solutions for fair *two-party* secure computation over the Bitcoin network. In contrast, in this work, we propose *formal security models* for fair computations, and construct fair secure computation and lottery in the *multiparty* setting. As far as fair two-party secure computation is concerned, although the goal of [3] and ours is the same, the means to achieve the goal are significantly different. Specifically, the protocols of [4, 3] directly works by building particular Bitcoin transactions (i.e., with no formal definitions of relevant functionalities). In the following, we provide a summary of other related works. (See also Appendix C.)

---

<sup>4</sup>Indeed, we can readily adapt our constructions to obtain the first multiparty solutions enjoying “legally enforceable” fairness [73].

<sup>5</sup>Contrast this with the gradual release mechanism which require security parameter number of rounds even when  $n = 2$ .

- *Fairness in standard secure computation.* Fair two party coin tossing was shown to be impossible in [35]. Completely fair secure computation for restricted classes of functions was shown in [53, 5], while partially fair secure computation for all functions were constructed in [56, 14]. Complete primitives for fairness were extensively studied in [54].
- *Gradual release mechanisms.* Starting from early works [13, 50], gradual release mechanism have been employed to solve the problem of fair exchange in several settings [22, 42, 46]. A good survey of this area can be found in [90]. A formal treatment of gradual release mechanisms can also be found in [45].
- *Optimistic model.* There has been a huge body of work starting from [9, 8, 18] that deals with optimistic models for fair exchange (e.g., [69, 79, 43]). Optimistic models for secure computation was considered in [23]. [69] consider a model similar to ours where receiving payment in the event of breach of fairness is also considered fair. Optimistic fair exchange in the multiuser setting was studied in [7, 12, 40].
- *Incentivized computation and reputation systems.* The works of [52, 17] design a credit system where users are rewarded for good work and fined for cheating (assuming a trusted arbiter/supervisor in some settings). Fair secure computation with reputation systems was considered in [6]. It has been claimed that reputations systems find limited applicability because it is unclear how to define the reputation of new users [41].
- *Legally enforceable fairness.* Chen, Kudla, and Paterson [33] designed protocols for fair exchange of signatures in a model where signatures are validated only in a court-of-law. Following this, Lindell [73] showed how to construct legally enforceable fairness in the two party secure computation where parties have access to a trusted bank (or a court of law).
- *E-cash and cryptographic currencies.* Starting from Chaum’s work [31], there has been a lot of work proposing use of e-cash for protecting privacy of financial transactions. (See [94, 68, 17, 24, 25] and references therein.) In contrast with traditional e-cash that requires a central trusted bank, recent years have seen the emergence of new decentralized systems such as Bitcoin [84] that allow anonymous spending of coins.
- *Rational adversaries.* There is a long line of work that attempts to unify cryptography and game theory. Applications of cryptography to game theory include the works of [39, 1, 2, 62]. More directly related to secure computation are the works of [75, 57, 58, 55, 60, 88, 71]. Protocols for multiparty lottery are also designed in [70, 4, 11].

## 2 Models and Definitions

Before we begin, we note that our formalization is heavily inspired by prior formalizations in settings similar to ours [73, 45]. Let  $n$  denote the number of parties and  $t$  (resp.  $h$ ) denote the number of corrupted (resp. honest) parties. We consider settings where  $t < n$ .<sup>6</sup> In our setting we are interested in dealing with non-standard commodities which we call “coins,” that cannot be directly incorporated in standard definitions of secure computation.

**Coins.** In this paper, we define *coins* as *atomic entities that are fungible and cannot be duplicated*. In particular, we assume coins have the following properties: (1) the owner of a coin is simply the party that possesses it, and further it is guaranteed that *no other party can possess that coin simultaneously*, and (2) coins can be freely transferred from a sender to a receiver (i.e, the sender is no longer the owner of the item while the receiver becomes the new owner of the item), and further, the validity of a received coin can be immediately checked and confirmed. Note we assume that *each coin is perfectly indistinguishable from one another*. Further we assume that each party has its own *wallet* and *safe*.<sup>7</sup> All its coins are distributed between its own wallet and its own safe.

Our definition of coin is intended to capture *physical/cryptographic currencies* contained in (individual) physical/cryptographic wallets. As such the above description of a coin does not capture digital cheques or financial contracts (i.e., those that need external parties such as banks or a court-of-law to validate them). However, we

---

<sup>6</sup>Note that even when  $t < n/2$ , it is not clear how to design a “fair” lottery simply because standard models do not deal with coins.

<sup>7</sup>The distinction between wallet and safe will become clear in the description of the ideal/real processes.

chose this definition to keep things simple, and more technically speaking, such a formalization would enable us to consider concurrent composition of protocols that deal with coins (in contrast with the formalization in [73]).

*Notation.* We use  $\text{coins}(x)$  to denote an item whose value is described by  $x \in \mathbb{N}$ . Suppose a party possesses  $\text{coins}(x_1)$  and receives  $\text{coins}(x_2)$  from another party, then we say it now possesses  $\text{coins}(x_1 + x_2)$ . Suppose a party possesses  $\text{coins}(x_1)$  and sends  $\text{coins}(x_2)$  to another party, then we say it now possesses  $\text{coins}(x_1 - x_2)$ .

**Model.** We will prove security of our protocols using the simulation paradigm. To keep things simple:

- Our protocols are designed in a hybrid model where parties have access to two *types* of ideal functionalities which we describe below. In the relevant hybrid model, our protocols will have *straightline* simulators, and thus we can hope for achieving standalone as well as universally composable (UC) security. We chose to provide UC-style definitions [27] of our ideal functionalities.
  - The first type of ideal functionalities are standard ideal functionalities used in secure computation literature (e.g., see  $\mathcal{F}_f$  described in Figure 4). These functionalities only provide security with agreement on abort [48, 51]. In particular, they do not provide the notion of fairness that we are interested in.
  - The second type of ideal functionalities are *special* ideal functionalities that deal with coins. These are the ideal functionalities that we will be interested in realizing. Note that only special ideal functionalities deal with coins.

Special ideal functionalities are denoted by  $\mathcal{F}_{\text{xxx}}^*$  (i.e., with superscript  $*$ ) to distinguish them notationally from standard ideal functionalities. We will be interested in *secure realization* of these functionalities.

- We work in the standard model of secure computation where parties are assumed to be connected with pairwise secure channels over a synchronous network (i.e., the computation proceeds in “rounds”). See [45, 65] on how to make the relevant modifications about synchrony assumptions in the UC-framework [27, 30].
- Our special ideal functionality  $\mathcal{F}_{\text{CR}}^*$  that idealizes Bitcoin transactions, is assumed to be aware of the round structure of the protocol. This choice is inspired by similar assumptions about the “wrapped functionalities” considered in [45].

*On the choice of UC-style definitions.* In practice, we expect parties to run variety of electronic transactions concurrently. A natural requirement for proving security would be to consider *universally composable* (UC) security which would in turn also enable modular design of protocols. Perhaps, the main drawback in considering UC security is the fact that to UC realize most (standard) functionalities one typically needs to assume the existence of a trusted setup [28, 29, 47]. To avoid this, one may design concurrently secure protocols based only on pure complexity-theoretic assumptions (see [80, 72] and references therein). Despite this, we chose to work in a UC-like framework (which we describe below) because we believe it enables simpler and cleaner abstraction and description of our ideal functionalities and our protocols. Also we argue that the trusted setup in UC is typically a one-time setup (as opposed to say the optimistic model where trusted help needs to be online).<sup>8</sup> Further, the standalone variant of our protocols require no such setup.

*Preliminaries.* A function  $\mu(\cdot)$  is negligible in  $\lambda$  if for every positive polynomial  $p(\cdot)$  and all sufficiently large  $\lambda$ 's it holds that  $\mu(\lambda) < 1/p(\lambda)$ . A probability ensemble  $X = \{X(a, \lambda)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$  is an infinite sequence of random variables indexed by  $a$  and  $\lambda \in \mathbb{N}$ . Two distribution ensembles  $X = \{X(a, \lambda)\}_{\lambda \in \mathbb{N}}$  and  $Y = \{Y(a, \lambda)\}_{\lambda \in \mathbb{N}}$  are said to be computationally indistinguishable, denoted  $X \stackrel{c}{\equiv} Y$  if for every non-uniform polynomial-time algorithm  $D$  there exists a negligible function  $\mu(\cdot)$  such that for every  $a \in \{0,1\}^*$ ,

$$|\Pr[D(X(a, \lambda)) = 1] - \Pr[D(Y(a, \lambda)) = 1]| \leq \mu(\lambda).$$

All parties are assumed to run in time polynomial in the security parameter  $\lambda$ . We follow standard definitions of secure computation [48]. Our main modification is now each party has its own wallet and safe, and further, the view of  $\mathcal{Z}$  contains the distribution of coins. We provide a succinct description of our model, which we call

---

<sup>8</sup>Also note, in practice, one may obtain heuristic UC security in the programmable random oracle model.

“security computation with coins” (SCC), highlighting the differences from standard secure computation. Before that we describe the distinction between wallets and safes.

*Wallets vs. safes.* Recall that in standard models each party is modeled as an interactive Turing machine. For our purposes, we need to augment the model by providing each party with its own wallet and safe. We allow each party’s wallet to be arbitrarily modified by the distinguisher  $\mathcal{Z}$  (aka environment). However, honest parties’ safes are out of  $\mathcal{Z}$ ’s control. This is meant to reflect honest behavior in situations where the party has no coins left to participate in a protocol. We require honest parties to simply not participate in such situations. In other words, in order to participate in a protocol, an honest party first locks the required number of coins (specified by the protocol) in its safe. During the course of a protocol, the honest party may gain coins (e.g., by receiving a penalty), or may lose coins (e.g., in a lottery). These gains and losses affect the content of the safes and not the wallets. Finally, at the end of the protocol, the honest party releases the coins associated with that protocol (including new gains) into the wallet. Note on the other hand, we give the adversary complete control over a corrupt party’s wallet *and* safe.

**Secure computation with coins (SCC security).** We now describe the ideal/real processes for SCC. The order of activations is the same as in UC, and in particular,  $\mathcal{Z}$  is activated first. In each activation of  $\mathcal{Z}$ , in addition to choosing (both honest and corrupt) parties’ inputs (as in standard UC),  $\mathcal{Z}$  also initializes each party’s wallet with some number of coins and may activate the hybrid (resp. ideal) adversary  $\mathcal{A}$  (resp.  $\mathcal{S}$ ). In every subsequent activation,  $\mathcal{Z}$  may read and/or modify (i.e., add coins to or retrieve coins from)<sup>9</sup> the contents of the wallet (but *not* the safe) of each honest party. Further,  $\mathcal{Z}$  may also read each honest party’s local output tapes, and may write information on its input tape. In the hybrid (resp. ideal) process, the adversary  $\mathcal{A}$  (resp.  $\mathcal{S}$ ) has complete access to all tapes, wallets, and safes of a corrupt party. Note that, as in UC, the environment  $\mathcal{Z}$  will be an interactive distinguisher.

Let  $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)$  denote the output of environment  $\mathcal{Z}$  initialized with input  $z$  after interacting in the ideal process with ideal process adversary  $\mathcal{S}$  and (standard or special) ideal functionality  $\mathcal{G}_f$  on security parameter  $\lambda$ . Recall that our protocols will be run in a hybrid model where parties will have access to a (standard or special) ideal functionality  $\mathcal{G}_g$ . We denote the output of  $\mathcal{Z}$  after interacting in an execution of  $\pi$  in such a model with  $\mathcal{A}$  by  $\text{HYBRID}_{\pi,\mathcal{A},\mathcal{Z}}^g(\lambda, z)$ , where  $z$  denotes  $\mathcal{Z}$ ’s input. We are now ready to define what it means for a protocol to SCC realize a functionality.

**Definition 1.** Let  $n \in \mathbb{N}$ . Let  $\pi$  be a probabilistic polynomial-time  $n$ -party protocol and let  $\mathcal{G}_f$  be a probabilistic polynomial-time  $n$ -party (standard or special) ideal functionality. We say that  $\pi$  SCC realizes  $\mathcal{G}_f$  with abort in the  $\mathcal{G}_g$ -hybrid model (where  $\mathcal{G}_g$  is a standard or a special ideal functionality) if for every non-uniform probabilistic polynomial-time adversary  $\mathcal{A}$  attacking  $\pi$  there exists a non-uniform probabilistic polynomial-time adversary  $\mathcal{S}$  for the ideal model such that for every non-uniform probabilistic polynomial-time adversary  $\mathcal{Z}$ ,

$$\{\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*} \stackrel{c}{\equiv} \{\text{HYBRID}_{\pi,\mathcal{A},\mathcal{Z}}^g(\lambda, z)\}_{\lambda \in \mathbb{N}, z \in \{0,1\}^*}.$$

◇

We have not proven a composition theorem for our definition (although we believe our model should in principle allow composition analogous to the UC composition theorem [27]). For the results in this paper, we only need to *assume* that the Bitcoin protocol realizing  $\mathcal{F}_{\text{CR}}^*$  is concurrently composable. Other than this, we require only standard sequential composition [26]. We stress that our protocols enjoy straightline simulation (both in the way coins and cryptographic primitives are handled), and thus they may be adaptable to a concurrent setting. Finally, we note that we consider only static corruptions.

Next, we define the security notion we wish to realize for fair secure computation and for fair lottery.

**Definition 2.** Let  $\pi$  be a protocol and  $f$  be a multiparty functionality. We say that  $\pi$  securely computes  $f$  with penalties if  $\pi$  SCC-realizes the functionality  $\mathcal{F}_f^*$  according to Definition 1.

---

<sup>9</sup>I.e., we implicitly give  $\mathcal{Z}$  the power to create new coins.

Functionality  $\mathcal{F}_{\text{CR}}^*$

$\mathcal{F}_{\text{CR}}^*$  with session identifier  $sid$ , running with parties  $P_1, \dots, P_n$ , a parameter  $1^\lambda$ , and an ideal adversary  $\mathcal{S}$  proceeds as follows:

- *Deposit phase.* Upon receiving the tuple  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, \text{coins}(x))$  from  $P_s$ , record the message  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$  and send it to all parties. Ignore any future deposit messages with the same  $ssid$  from  $P_s$  to  $P_r$ .
- *Claim phase.* In round  $\tau$ , upon receiving  $(\text{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, x, w)$  from  $P_r$ , check if (1) a tuple  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$  was recorded, and (2) if  $\phi_{s,r}(w) = 1$ . If both checks pass, send  $(\text{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, x, w)$  to all parties, send  $(\text{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, \text{coins}(x))$  to  $P_r$ , and delete the record  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$ .
- *Refund phase:* In round  $\tau + 1$ , if the record  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$  was not deleted, then send  $(\text{refund}, sid, ssid, s, r, \phi_{s,r}, \tau, \text{coins}(x))$  to  $P_s$ , and delete the record  $(\text{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$ .

Figure 1: The special ideal functionality  $\mathcal{F}_{\text{CR}}^*$ .

**Definition 3.** Let  $\pi$  be a protocol. We say that  $\pi$  is a secure lottery with penalties if  $\pi$  SCC-realizes the functionality  $\mathcal{F}_{\text{lot}}^*$  according to Definition 1.

## 2.1 Special ideal functionalities

**Ideal functionality  $\mathcal{F}_{\text{CR}}^*$ .** This is our main special ideal functionality and will serve as a building block for securely realizing more complex special functionalities. (See Figure 1 for a formal description.) At a very basic level,  $\mathcal{F}_{\text{CR}}^*$  allows a sender  $P_s$  to *conditionally* send  $\text{coins}(x)$  to a receiver  $P_r$ . The condition is formalized as the revelation of a satisfying assignment (i.e., witness) for a sender-specified circuit  $\phi_{s,r}$  (i.e., relation). Further, there is a “time” bound, formalized as a round number  $\tau$ , within which  $P_r$  has to act in order to claim the coins. An important property that we wish to stress is that the satisfying witness is made *public* by  $\mathcal{F}_{\text{CR}}^*$ .

The importance of the above functionality is a highly efficient realization via *Bitcoin* that requires only two transactions to be made on the network. The Bitcoin realization is shown in Figure 9. In the Bitcoin realizations of the ideal functionalities, sending a message with  $\text{coins}(x)$  corresponds to broadcasting a transaction to the Bitcoin network, and waiting according to some time parameter until there is enough confidence that the transaction will not be reversed.

**Secure computation with penalties.** Loosely speaking, our notion of fair secure computation guarantees:

- An honest party never has to pay any penalty.
- If a party aborts after learning the output and does not deliver output to honest parties, then *every* honest party is compensated.

These guarantees are exactly captured in our description of the ideal functionality  $\mathcal{F}_f^*$  for secure computation with penalties in Figure 2. We elaborate more on the definition of the ideal functionality  $\mathcal{F}_f^*$  below.

**Ideal functionality  $\mathcal{F}_f^*$ .** In the first phase, the functionality  $\mathcal{F}_f^*$  receives inputs for  $f$  from all parties. In addition,  $\mathcal{F}_f^*$  allows the ideal world adversary  $\mathcal{S}$  to deposit some coins which may be used to compensate honest parties if  $\mathcal{S}$  aborts after receiving the outputs. Note that an honest party makes a fixed deposit  $\text{coins}(d)$  in the input phase.<sup>10,11</sup> Then, in the output phase,  $\mathcal{F}_f^*$  returns the deposit made by honest parties back to them. If insufficient

<sup>10</sup>Ideally, we wouldn’t want an honest party to deposit any coins, but we impose this requirement for technical reasons.

<sup>11</sup>To keep the definitions simple (here and in the following), we omitted details involving obvious checks that will be performed to ensure parties provide correct inputs to the ideal functionality, including (1) checks that the provided coins are valid, and (2) deposit amounts are consistent across all parties. If checks fail, then the ideal functionality simply informs all parties and terminates the session.

### Functionality $\mathcal{F}_f^*$

$\mathcal{F}_f^*$  with session identifier  $sid$  running with parties  $P_1, \dots, P_n$ , a parameter  $1^\lambda$ , and an ideal adversary  $\mathcal{S}$  that corrupts parties  $\{P_s\}_{s \in C}$  proceeds as follows: Let  $H = [n] \setminus C$  and  $h = |H|$ . Let  $d$  be a parameter representing the safety deposit, and let  $q$  denote the penalty amount.

- *Input phase:* Wait to receive a message  $(\text{input}, sid, ssid, r, y_r, \text{coins}(d))$  from  $P_r$  for all  $r \in H$ . Then wait to receive a message  $(\text{input}, sid, ssid, \{y_s\}_{s \in C}, H', \text{coins}(h'q))$  from  $\mathcal{S}$  where  $h' = |H'|$ .
- *Output phase:*
  - Send  $(\text{return}, sid, ssid, \text{coins}(d))$  to each  $P_r$  for  $r \in H$ .
  - Compute  $(z_1, \dots, z_n) \leftarrow f(y_1, \dots, y_n)$ .
    - If  $h' = 0$ , then send message  $(\text{output}, sid, ssid, z_r)$  to  $P_r$  for  $r \in [n]$ , and terminate.
    - If  $0 < h' < h$ , then send  $(\text{extra}, sid, ssid, \text{coins}(q))$  to  $P_r$  for each  $r \in H'$ , and terminate.
    - If  $h' = h$ , then send message  $(\text{output}, sid, ssid, \{z_s\}_{s \in C})$  to  $\mathcal{S}$ .
  - If  $\mathcal{S}$  returns  $(\text{continue}, sid, ssid, H'')$ , then send  $(\text{output}, sid, ssid, z_r)$  to  $P_r$  for all  $r \in H$ , and send  $(\text{payback}, sid, ssid, \text{coins}((h - h'')q))$  to  $\mathcal{S}$  where  $h'' = |H''|$ , and send  $(\text{extrapay}, sid, ssid, \text{coins}(q))$  to  $P_r$  for each  $r \in H''$ .
  - Else if  $\mathcal{S}$  returns  $(\text{abort}, sid, ssid)$ , send  $(\text{penalty}, sid, ssid, \text{coins}(q))$  to  $P_r$  for all  $r \in H$ .

Figure 2: The special ideal functionality  $\mathcal{F}_f^*$  for secure computation with penalties.

number of coins are deposited, then  $\mathcal{S}$  does not obtain the output, yet may potentially pay penalty to some subset  $H'$  of the honest parties. If  $\mathcal{S}$  deposited sufficient number of coins, then it gets a chance to look at the output and then decide to continue delivering output to all parties (and further pay an additional “penalty” to some subset  $H''$ ), or just abort, in which case *all* honest parties are compensated using the penalty deposited by  $\mathcal{S}$ .

**Secure lottery with penalties.** Loosely speaking, our notion of fair lottery guarantees the following:

- An honest party never has to pay any penalty.
- The lottery winner has to be chosen uniformly at random.
- If a party aborts *after* learning whether or not it won the lottery without disclosing this information to honest parties, then every honest party is compensated.

These guarantees are exactly captured in our description of the ideal functionality  $\mathcal{F}_{\text{lot}}^*$  for secure lottery with penalties in Figure 3. We elaborate more on the definition of the ideal functionality  $\mathcal{F}_{\text{lot}}^*$  below.

**Ideal functionality  $\mathcal{F}_{\text{lot}}^*$ .** The high level idea behind the design of  $\mathcal{F}_{\text{lot}}^*$  is the same as that for  $\mathcal{F}_f^*$ . The main distinction is that now the functionality has to ensure that the lottery is conducted properly, in the sense that all parties pay their fair share of the lottery prize (i.e.,  $\text{coins}(q/n)$ ). Thus we require that each honest party makes a fixed lottery deposit  $\text{coins}(d)$  with  $d \geq q/n$ . Then, in the second phase, as was the case with  $\mathcal{F}_f^*$ , the ideal functionality  $\mathcal{F}_{\text{lot}}^*$  allows  $\mathcal{S}$  to learn the outcome of the lottery only if it made a sufficient penalty deposit (i.e.,  $\text{coins}(hq + (tq/n))$ ). As before, if  $\mathcal{S}$  decides to abort, then *all* honest parties are compensated using the penalty deposited by  $\mathcal{S}$  in addition to getting their lottery deposits back. (I.e., effectively, *every* honest party wins the lottery!)

**Remarks.** At first glance, it may appear that the sets  $H', H''$  (resp.  $H', \tilde{H}', H''$ ) in the definition of  $\mathcal{F}_f^*$  (resp.  $\mathcal{F}_{\text{lot}}^*$ ) are somewhat unnatural. We stress that we require specification of these sets in the ideal functionalities in order to ensure that we can prove that our protocols securely realize these functionalities. We also stress that it is plausible that a different security definition (cf. Definitions 2, 3) or a different protocol construction may satisfy more “natural” formulations of  $\mathcal{F}_f^*$  and  $\mathcal{F}_{\text{lot}}^*$ . We leave this for future work.

Functionality  $\mathcal{F}_{\text{lot}}^*$

$\mathcal{F}_{\text{lot}}^*$  with session identifier  $sid$  running with parties  $P_1, \dots, P_n$ , a parameter  $1^\lambda$ , and an ideal adversary  $\mathcal{S}$  that corrupts parties  $\{P_s\}_{s \in C}$  proceeds as follows: Let  $H = [n] \setminus C$  and  $h = |H|$  and  $t = |C|$ . Let  $d$  be a parameter representing the safety deposit, and let  $q$  be the value of the lottery prize (note:  $q$  is also the penalty amount). We assume  $d \geq q/n$ .

- *Input phase:* Wait to receive a message  $(\text{input}, sid, ssid, r, \text{coins}(d))$  from  $P_r$  for all  $r \in H$ . Then wait to receive a message  $(\text{input}, sid, ssid, \{y_s\}_{s \in C}, H', \text{coins}(h'q + (tq/n)))$  from  $\mathcal{S}$  where  $h' = |H'|$ .
- *Output phase:* Choose  $r^* \leftarrow_R \{1, \dots, n\}$ .
  - If  $h' = 0$ , then send message  $(\text{output}, sid, ssid, r^*)$  to  $P_r$  for  $r \in [n]$ , and message  $(\text{return}, sid, ssid, \text{coins}(d - q/n))$  to each  $P_r$  for  $r \in H$ , and message  $(\text{prize}, sid, ssid, \text{coins}(q))$  to  $P_{r^*}$ , and terminate.
  - If  $0 < h' < h$ , then send  $(\text{extra}, sid, ssid, \text{coins}(q))$  to  $P_r$  for each  $r \in H'$ , and message  $(\text{return}, sid, ssid, \text{coins}(d))$  to each  $P_r$  for  $r \in H$ , and send  $(\text{sendback}, sid, ssid, \text{coins}(tq/n))$  to  $\mathcal{S}$ , and terminate.
  - If  $h' = h$ , then send message  $(\text{output}, sid, ssid, r^*)$  to  $\mathcal{S}$ .
  - If  $\mathcal{S}$  returns  $(\text{continue}, sid, ssid, \tilde{H}', H'')$ , then send message  $(\text{output}, sid, ssid, r^*)$  to  $P_r$  for  $r \in [n]$ , and message  $(\text{return}, sid, ssid, \text{coins}(d - q/n))$  to each  $P_r$  for  $r \in H$ , and message  $(\text{prize}, sid, ssid, \text{coins}(q))$  to  $P_{r^*}$ , and message  $(\text{extripay}_1, sid, ssid, \text{coins}(q))$  to  $P_r$  for  $r \in \tilde{H}'$ , and message  $(\text{extripay}_2, sid, ssid, \text{coins}(q/n))$  to  $P_r$  for  $r \in H''$ , and message  $(\text{payback}, sid, ssid, \text{coins}((h - \tilde{h}')q - h''q/n))$  to  $\mathcal{S}$  where  $\tilde{h}' = |\tilde{H}'|$  and  $h'' = |H''|$ , and terminate.
  - Else if  $\mathcal{S}$  returns  $(\text{abort}, sid, ssid)$ , send messages  $(\text{return}, sid, ssid, \text{coins}(d))$  and  $(\text{penalty}, sid, ssid, \text{coins}(q))$  to  $P_r$  for all  $r \in H$ , and messages  $(\text{sendback}, sid, ssid, \text{coins}(tq/n))$  to  $\mathcal{S}$ , and terminate.

Figure 3: The ideal functionality  $\mathcal{F}_{\text{lot}}^*$  for secure lottery with penalties.

### 3 Secure Multiparty Computation with Penalties

We design protocols for secure computation with penalties in a hybrid model with (1) a standard ideal functionality realizing an *augmented* version of the unfair underlying function we are interested in computing, and (2) a special ideal functionality denoted  $\mathcal{F}_{\text{rec}}^*$  (see Appendix A) that will enable us to provide fairness. In the following, we assume, without loss of generality, that  $f$  delivers the *same* output to all parties. For a function  $f$ , the corresponding augmented function  $\hat{f}$  performs secret sharing of the output of  $f$  using a variant of *non-malleable secret sharing scheme* that is both publicly verifiable and publicly reconstructible (in short, pubNMSS). Secure computation with penalties is then achieved via carrying out “fair reconstruction” for the pubNMSS scheme.<sup>12</sup>

First, we provide a high level description of the semantics of the pubNMSS scheme. The Share algorithm takes as input a secret  $u$ , and generates “tag-token” pairs  $\{(\text{Tag}_i, \text{Token}_i)\}_{i \in [n]}$ . Finally it outputs to each party  $P_i$  the  $i$ -th token  $\text{Token}_i$  and  $\text{AllTags} = (\text{Tag}_1, \dots, \text{Tag}_n)$ . Loosely speaking, the properties that we require from pubNMSS are (1) an adversary corrupting  $t < n$  parties does not learn any information about the secret unless all shares held by honest parties are disclosed (i.e., in particular,  $\text{AllTags}$  does not reveal any further information), and (2) for any  $j \in [n]$ , the adversary cannot reveal  $\text{Token}'_j \neq \text{Token}_j$  such that  $(\text{Tag}_j, \text{Token}'_j)$  is a valid tag-token pair. Since Share is evaluated inside a secure protocol, we are guaranteed honest generation of tags and tokens. Given this, a natural candidate for a pubNMSS scheme can be obtained via *commitments* that are binding for *honest* sender (exactly as in [44]) and are equivocal. Instantiating a variant of the Naor commitment scheme [85] as done in [44], we obtain a construction of a pubNMSS scheme using only *one-way functions*. (See Section D.1 for more details.) We do not attempt to provide a formal definition of pubNMSS schemes. Rather,

<sup>12</sup>Our strategy is similar to the use of non-malleable secret sharing in [54] to construct complete primitives for fair secure computation in the *standard model*. In addition to working in a different model, the main difference is that here we explicitly require public verification and public reconstruction for the non-malleable secret sharing scheme. This requirement is in part motivated by the final Bitcoin realizations where validity of the shares need to be publicly verifiable (e.g., by miners) in order to successfully complete the transactions.

our approach here is to sketch a specific construction which essentially satisfies all our requirements outlined above. Given a secret  $u$ , we generate tag-token pairs in the following way:

- Perform an  $n$ -out-of- $n$  secret sharing of  $u$  to obtain  $u_1, \dots, u_n$ .
- To generate the  $i$ -th “tag-token” pair, apply the sender algorithm for a honest-binding commitment using randomness  $\omega_i$  to secret share  $u_i$  to obtain  $\text{com}_i$ , and set  $\text{Tag}_i = \text{com}_i$  and  $\text{Token}_i = (u_i, \omega_i)$ .

The reconstruction algorithm  $\text{Rec}$  takes as inputs  $(\text{AllTags}', \{\text{Token}'_i\}_{i \in [n]})$  and proceeds in a straightforward way. First, it checks if  $(\text{Tag}'_i, \text{Token}'_i = (u_i, \omega_i))$  is a valid tag-token pair (i.e., if  $\text{Token}'_i$  is a valid decommitment for  $\text{Tag}'_i$ ) for every  $i \in [n]$ . Next, if the check passes, then it outputs  $u' = \bigoplus_{\ell \in [n]} u'_\ell$ , else it outputs  $\perp$ .

In the following we show how to perform “fair reconstruction” for the scheme described above.

### 3.1 Fair Reconstruction

Loosely speaking, our notion of fair reconstruction guarantees the following:

- An honest party never has to pay any penalty.
- If the adversary reconstructs the secret, but an honest party cannot, then the honest party is compensated.

These guarantees are captured in our definition of  $\mathcal{F}_{\text{rec}}^*$  (see Appendix A). In this section, we show how to design a protocol for  $\mathcal{F}_{\text{rec}}^*$  in the  $\mathcal{F}_{\text{CR}}^*$ -hybrid model. We present our solution in a step-by-step manner. We first consider the two-party setting, and then show how to generalize to the multiparty case.

**Notation.** As discussed before, we assume that the secret has been shared using pubNMSS, i.e., each party  $P_i$  now has  $\text{AllTags}$  and its own token  $\text{Token}_i$ . Once a party learns all the tokens, then it can reconstruct the secret. On the other hand, even if one token is not revealed, then the secret is hidden. We use  $T_i$  as shorthand to denote  $\text{Token}_i$ . A sender  $P_s$  may use (a set of) tags to specify a  $\mathcal{F}_{\text{CR}}^*$  transaction with the guarantee that (except with negligible probability) its deposit can be claimed by a receiver  $P_r$  only if it produces the corresponding (set of) tokens. (More precisely, this is captured via the relation  $\phi_{s,r}$  specified by  $P_s$ ). In the following, we use  $P_1 \xrightarrow[q,\tau]{T} P_2$  to represent a  $\mathcal{F}_{\text{CR}}^*$  deposit transaction made by  $P_1$  with coins( $q$ ) which can be claimed by  $P_2$  in round  $\tau$  only if it produces token  $T$ , and if  $P_2$  does not claim the transaction, then  $P_1$  gets coins( $q$ ) refunded back after round  $\tau$ . We use  $\tau_1, \dots, \tau_n$  to denote round numbers. In order to keep the presentation simple and easy to follow, we avoid specifying the exact round numbers, and instead only specify constraints, e.g.,  $\tau_1 < \tau_2$ ,

**Naïve approach.** Consider the following protocol for the 2-party setting, where both parties hold  $\{\text{Tag}(a_1), \text{Tag}(a_2)\}$ , and additionally,  $P_1$  holds  $T_1 = \text{Token}(a_1)$  while  $P_2$  holds  $T_2 = \text{Token}(a_2)$ .<sup>13</sup> (I.e., the secret to be reconstructed is  $a_1 \oplus a_2$ , and parties  $P_1, P_2$  possesses  $a_1, a_2$  respectively). A first approach to achieving fair reconstruction would be a protocol like this.

$$P_1 \xrightarrow[q,\tau]{T_2} P_2 \quad (1)$$

$$P_2 \xrightarrow[q,\tau]{T_1} P_1 \quad (2)$$

If both parties make the deposit transactions, then an honest party can always use its token to claim the penalty deposited by the other party. Further, even if the malicious party aborts before revealing its token, the honest party gets its deposit back (on top of the claimed penalty). Therefore, the above approach appears to work at first glance. Unfortunately, this protocol is susceptible to an attack by malicious  $P_2$  that waits for  $P_1$  to make the deposit transaction (i.e., Step 1), after which  $P_2$  simply does not execute Step 2. Then,  $P_2$  claims the first transaction, and obtains  $q$  while  $P_1$  obtains  $T_2$  (and thus completes the reconstruction). Effectively, this means that an honest  $P_1$  has “paid”  $q$  (dubbed “penalty” amount) to learn  $P_2$ ’s token. Clearly, this violates our first requirement, namely that an honest party should never have to pay any penalty.

---

<sup>13</sup>We point out this notation is somewhat inaccurate. Specifically  $\text{Token}(a)$  should also include some randomness in addition to  $a$ .

**Our 2-party solution.** We circumvent the attack described above by imposing an additional constraint that  $P_2$  has to meet in order to claim the first transaction. Specifically,  $P_2$  has to reveal  $T_1$  in addition to  $T_2$  in order to claim the transaction made by  $P_1$ . Let  $\tau_1, \tau_2$  be such that  $\tau_1 < \tau_2$ . Our modified protocol looks like this.

$$P_1 \xrightarrow[q, \tau_2]{T_1 \wedge T_2} P_2 \quad (1)$$

$$P_2 \xrightarrow[q, \tau_1]{T_1} P_1 \quad (2)$$

The deposits are claimed in the *reverse* direction, namely the deposit in Step 2 is claimed by  $P_1$  by revealing  $T_1$ . This value of  $T_1$  can then be used by  $P_2$  (along with  $T_2$ ) to claim the deposit in Step 1. Clearly, if both parties are honest, then no party pays any penalty, and both parties obtain the tokens.

As before, if both parties make their respective deposit transactions, then the above protocol guarantees fair reconstruction. Now suppose that a malicious  $P_2$  aborts after  $P_1$  made its deposit. Since  $P_2$  never made its deposit, an honest  $P_1$  will not reveal  $T_1$ . Therefore,  $P_2$  cannot claim  $P_1$ 's deposit since it does not know  $T_1$ . In other words, the attack launched by  $P_2$  on the naïve solution no longer works. On the other hand, suppose  $P_1$  was malicious, and it aborted without making its deposit. In this case, since Step 1 did not happen, an honest  $P_2$  will not execute Step 2 (i.e., does not make a deposit). Therefore, neither party learns the output (nor loses its deposit).

**A naïve generalization.** Consider the 3-party setting, where all parties hold  $\{\text{Tag}(a_1), \text{Tag}(a_2), \text{Tag}(a_3)\}$ , and additionally  $P_1, P_2, P_3$  respectively hold tokens  $T_1 = \text{Token}(a_1)$ ,  $T_2 = \text{Token}(a_2)$ ,  $T_3 = \text{Token}(a_3)$ . Let  $\tau_1, \tau_2, \tau_3$  be such that  $\tau_1 < \tau_2 < \tau_3$ . A natural generalization of our 2-party solution for 3-party fair exchange would look like this.

$$P_1 \xrightarrow[q, \tau_3]{T_1 \wedge T_2 \wedge T_3} P_2 \quad (1)$$

$$P_2 \xrightarrow[q, \tau_2]{T_1 \wedge T_3} P_3 \quad (2)$$

$$P_3 \xrightarrow[q, \tau_1]{T_1} P_1 \quad (3)$$

Unfortunately, the above protocol is susceptible to an attack by a malicious coalition of  $P_1$  and  $P_2$ . Suppose all 3 parties make their deposits as described above. Then,  $P_1$  claims the deposit made by  $P_3$  by revealing  $T_1$ . Next, (honest)  $P_3$  claims the deposit made by  $P_2$  by revealing  $T_3$  along with  $T_1$  (which was obtained when  $P_1$  claimed its transaction). Note that at this point, the malicious coalition of  $P_1$  and  $P_2$  obtains all tokens  $T_1, T_2, T_3$ , while the honest party  $P_3$  still does not know  $T_2$ . Now, a malicious  $P_2$  simply aborts, and in particular does not claim  $P_1$ 's transaction. Note that  $P_1$ 's deposit will be refunded back after round  $\tau_3$ . Summarizing,  $P_3$  does not obtain token  $T_2$ , while  $P_1, P_2$  obtain all tokens, complete the reconstruction, and yet do not pay any penalty to  $P_3$ . In other words, our second requirement for fair reconstruction (namely that if an adversary reconstructs the secret, but an honest party cannot, then the honest parties is compensated), has been violated.

**Our 3-party solution.** Our protocol proceeds as follows. In Step 1, party  $P_1$  makes a deposit transaction that can be claimed by  $P_3$  as long as it produces  $T_1, T_2$ , and  $T_3$ . In Step 2, party  $P_2$  makes a similar deposit to  $P_3$ . Next, in Step 3, party  $P_3$  makes a deposit to  $P_2$  that can be claimed by  $P_2$  by revealing  $T_1$  and  $T_2$ . This deposit is made to *twice the penalty amount*, i.e.,  $2q$ . Finally, in Step 4, party  $P_2$  makes a deposit to  $P_1$  that can be claimed by  $P_1$  by revealing  $T_1$ . Let  $\tau_1, \tau_2, \tau_3$  be such that  $\tau_1 < \tau_2 < \tau_3$ . Pictorially, our solution for 3-party fair reconstruction looks like this.

$$P_1 \xrightarrow[q, \tau_3]{T_1 \wedge T_2 \wedge T_3} P_3 \quad (1)$$

$$P_2 \xrightarrow[q, \tau_3]{T_1 \wedge T_2 \wedge T_3} P_3 \quad (2)$$

$$P_3 \xrightarrow[2q, \tau_2]{T_1 \wedge T_2} P_2 \quad (3)$$

$$P_2 \xrightarrow[q, \tau_1]{T_1} P_1 \quad (4)$$

As in the 2-party case, the deposits are claimed in the reverse direction. Note that the tokens required to claim the deposit in Step  $i$  consist of the token possessed by the recipient of the Step  $i$  deposit plus the tokens required to claim the deposit in Step  $(i + 1)$ . Therefore, if deposit in Step  $(i + 1)$  is claimed, then the deposit in Step  $i$  can always be claimed. In particular, the above holds even if a deposit in Step  $j$  for  $j > i + 1$  was not claimed by a possibly corrupt party. Further, it can be verified that if all parties behave honestly, then the amount deposited equals the amount claimed.

Next, note that in our construction,  $P_3$  is the first party that obtains all tokens. If only  $P_3$  aborts (i.e., does not claim deposits from  $P_1$  and  $P_2$ ), then honest parties do not obtain  $P_3$ 's tokens, but both  $P_1$  and  $P_2$  get their deposits (in Step 1 and Step 2) refunded after round  $\tau_3$ . It can be verified that at this point, both  $P_1$  and  $P_3$  have been compensated by amount  $q$ .

**Multiparty fair reconstruction via the “ladder” construction.** In the general case, we will ask parties to make deposits in two phases. In the first phase, parties  $P_1, \dots, P_n$  simultaneously make a deposit of coins( $q$ ) to recipient  $P_n$  that can be claimed only if tokens  $T_1, \dots, T_n$  are produced by  $P_n$ . We call these deposits roof deposits. Then, in the second phase, each  $P_{s+1}$  makes a deposit of coins( $s \cdot q$ ) to recipient  $P_s$  that can be claimed only if tokens  $T_1, \dots, T_s$  are produced by  $P_s$ . These deposits are called the ladder deposits. For reasons that will be clear later, it is important that  $P_{s+1}$  makes its ladder deposit only if for all  $r > s + 1$ , party  $P_r$  already made its ladder deposit. We present a pictorial description of the deposit phase of the  $n$ -party protocol in Figure 6.

We deal with aborts in the deposit phase in the following way. If a corrupt party does not make the roof deposit it is supposed to make, then all parties get their roof deposits refunded following which they terminate the protocol. On the other hand, if a corrupt party  $P_r$  fails to make the ladder deposit it is supposed to make, then for all  $s < r$ , party  $P_s$  does not make its ladder deposit at all, while for all  $s > r$ , party  $P_s$  continues to wait until a designated round to see whether its ladder deposit is claimed (and in particular, does not terminate the protocol immediately).

As before, the deposits are claimed in the reverse direction. Note that the tokens required to claim the  $i$ -th ladder deposit consist of tokens possessed by the recipient of the  $i$ -th ladder deposit plus the tokens required to claim the  $(i + 1)$ -th ladder deposit (for  $i + 1 < n$ ). Therefore, if the  $(i + 1)$ -th ladder deposit is claimed, then the  $i$ -th ladder deposit can *always* be claimed. In particular, the above holds even if for some  $j > i + 1$ , (1) the  $j$ -th ladder deposit was not claimed by a possibly corrupt party, or (2) the  $j$ -th ladder deposit was not even made (which indeed is the reason why we require parties that have made their ladder deposit to wait even if a subsequent ladder deposit was not made). Further, it can be verified that if all parties behave honestly, then across all roof and ladder deposits, the amount deposited equals the amount claimed. We present a formal description of the protocol in the  $\mathcal{F}_{\text{CR}}^*$ -hybrid model in Figure 7. See Appendix D for technical details and a proof of security of the following theorem. Since  $\mathcal{F}_{\text{OT}}$ , the ideal functionality for oblivious transfer, is sufficient [64, 66] to compute any standard ideal functionality (and further implies one-way functions), we have the following theorem:

**Theorem 1.** *Assuming the existence of one-way functions, for every  $n$ -party functionality  $f$  there exists a protocol that securely computes  $f$  with penalties in the  $(\mathcal{F}_{\text{OT}}, \mathcal{F}_{\text{CR}}^*)$ -hybrid model. Further, the protocol requires  $O(n)$  rounds, a total of  $O(n)$  calls to  $\mathcal{F}_{\text{CR}}^*$ , and each party deposits  $n$  times the penalty amount.*

Somewhat surprisingly, minor modifications to the above protocol leads us to a construction for secure lotteries with penalties. Due to space limitations, the extension and the construction are described in Appendix E.

**Acknowledgments.** We would like to thank Yuval Ishai for many useful discussions. The first author thanks Eli Ben-Sasson for his encouragement and support.

## References

- [1] Joël Alwen, Jonathan Katz, Yehuda Lindell, Giuseppe Persiano, Abhi Shelat, and Ivan Visconti. Collusion-free multiparty computation in the mediated model. In Shai Halevi, editor, *Advances in Cryptology — Crypto 2009*, volume 5677 of *LNCS*, pages 524–540. Springer, 2009.
- [2] Joël Alwen, Jonathan Katz, Ueli Maurer, and Vassilis Zikas. Collusion-preserving computation. In *Crypto*, pages 124–143, 2012.
- [3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via the bitcoin deposits. In *ePrint 2013/837*, 2013.
- [4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.
- [5] Gilad Asharov. Towards characterizing complete fairness in secure two-party computation. In *To appear in Theory of Cryptography Conference*, 2014.
- [6] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *Asiacrypt* (2), pages 201–220, 2013.
- [7] N. Asokan, B. Baum-Waidner, M. Schunter, and M. Waidner. Optimistic synchronous multiparty contract signing, 1998. Technical Report RZ3089, IBM Research Report.
- [8] N. Asokan, V. Shoup, and M. Waidner. Optimistic protocols for fair exchange. In *ACM CCS*, pages 7–17, 1997.
- [9] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures (extended abstract). In Kaisa Nyberg, editor, *Advances in Cryptology — Eurocrypt ’98*, volume 1403 of *LNCS*, pages 591–606. Springer, 1998.
- [10] G. Avoine and S. Vaudenay. Optimistic fair exchange based on publicly verifiable secret sharing. In *ACISP*, pages 74–85, 2004.
- [11] A. Back and I. Bentov. Note on fair coin toss via bitcoin. <https://www.cs.technion.ac.il/%7Eidddo/coinossBitcoin.pdf>, 2013. <http://arxiv.org/abs/1402.3698>.
- [12] B. Baum-Waidner and M. Waidner. Optimistic asynchronous multiparty contract signing, 1998. Technical Report RZ3078, IBM Research Report.
- [13] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 468–473. IEEE, October / November 1989.
- [14] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov.  $1/p$ -Secure multiparty computation without honest majority and the best of both worlds. In *Advances in Cryptology — Crypto 2011*, volume 6841 of *LNCS*, pages 277–296. Springer, 2011.
- [15] Amos Beimel, Eran Omri, and Ilan Orlov. Protocols for multiparty coin toss with dishonest majority. In *Advances in Cryptology — Crypto 2010*, volume 6223 of *LNCS*, pages 538–557. Springer, 2010.
- [16] M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Kupcu, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *Proc. of WPES*, 2007.
- [17] Mira Belenkiy, Melissa Chase, C. Christopher Erway, John Jannotti, Alptekin Kupcu, and Anna Lysyanskaya. Incentivizing outsourced computation. In *NetEcon*, pages 85–90, 2008.
- [18] Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ron Rivest. A fair protocol for signing contracts (extended abstract). In *ICALP*, pages 43–52, 1985.
- [19] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10. ACM Press, May 1988.
- [20] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Practical decentralized anonymous e-cash from bitcoin. 2014. *IEEE Security and Privacy*.
- [21] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake, 2014. Preprint.
- [22] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.

- [23] Christian Cachin and Jan Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 93–111. Springer, 2000.
- [24] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology — Eurocrypt 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, 2005.
- [25] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security & Privacy*, pages 101–115. IEEE, May 2007.
- [26] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
- [27] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, October 2001.
- [28] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — Crypto 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, 2001.
- [29] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology — Eurocrypt 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, 2003.
- [30] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503. ACM Press, May 2002.
- [31] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology — Crypto '82*, pages 199–203. Plenum Press, 1983.
- [32] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19. ACM Press, May 1988.
- [33] Liquan Chen, Caroline Kudla, and Kenneth G. Paterson. Concurrent signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology — Eurocrypt 2004*, volume 3027 of *LNCS*, pages 287–305. Springer, 2004.
- [34] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin - (short paper). In *Financial Cryptography*, pages 390–398, 2012.
- [35] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.
- [36] Dana Dachman-Soled, Yehuda Lindell, Mohammad Mahmoody, and Tal Malkin. On the black-box complexity of optimally-fair coin tossing. In *8th Theory of Cryptography Conference — TCC 2011*, volume 6597 of *LNCS*, pages 450–467. Springer, 2011.
- [37] Dana Dachman-Soled, Mahmood Mahmoody, and Tal Malkin. Can optimally-fair coin tossing be based on one-way functions? In *To appear in Theory of Cryptography Conference*, 2014.
- [38] Y. Dodis and L. Reyzin. Breaking and repairing optimistic fair exchange from podc 2003. In *ACM Workshop on Digital Rights Management*, pages 47–54, 2003.
- [39] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In Mihir Bellare, editor, *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 112–130. Springer, 2000.
- [40] Yevgeniy Dodis, Pil Joong Lee, and Dae Hyun Yum. Optimistic fair exchange in a multi-user setting. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *10th Intl. Conference on Theory and Practice of Public Key Cryptography(PKC 2007)*, volume 4450 of *LNCS*, pages 118–133. Springer, April 2007.
- [41] E. Friedman and P. Resnick. The social cost of cheap pseudonyms. In *Journal of Economics and Management Strategy*, pages 173–199, 2000.
- [42] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In Matt Blaze, editor, *Financial Cryptography and Data Security 2002*, volume 2357 of *LNCS*, pages 168–182. Springer, March 2002.
- [43] Juan A. Garay, Markus Jakobsson, and Philip D. MacKenzie. Abuse-free optimistic contract signing. In Michael J. Wiener, editor, *Advances in Cryptology — Crypto '99*, volume 1666 of *LNCS*, pages 449–466. Springer, 1999.
- [44] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. pages 179–186. ACM Press, 2011.
- [45] Juan A. Garay, Philip D. MacKenzie, Manoj Prabhakaran, and Ke Yang. Resource fairness and composability of cryptographic protocols. In Shai Halevi and Tal Rabin, editors, *3rd Theory of Cryptography Conference — TCC 2006*, volume 3876 of *LNCS*, pages 404–428. Springer, March 2006.
- [46] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In Rebecca Wright, editor, *Financial Cryptography and Data Security 2003*, volume 2742 of *LNCS*, pages 190–207. Springer, January 2003.
- [47] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do UC. In *8th Theory of Cryptography Conference — TCC 2011*, volume 6597 of *LNCS*, pages 311–328. Springer, 2011.

- [48] Oded Goldreich. Foundations of cryptography - volume 2, basic applications. 2004.
- [49] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, May 1987.
- [50] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology — Crypto '90*, volume 537 of *LNCS*, pages 77–93. Springer, 1991.
- [51] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.
- [52] Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In David Naccache, editor, *Cryptographers' Track — RSA 2001*, volume 2020 of *LNCS*, pages 425–440. Springer, April 2001.
- [53] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–422. ACM Press, May 2008.
- [54] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In *7th Theory of Cryptography Conference — TCC 2010*, volume 5978 of *LNCS*, pages 91–108. Springer, 2010.
- [55] S. Dov Gordon and Jonathan Katz. Rational secret sharing, revisited. In Roberto De Prisco and Moti Yung, editors, *5th Intl. Conf. on Security and Cryptography for Networks*, volume 4116 of *LNCS*, pages 229–241. Springer, September 2006.
- [56] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In *Advances in Cryptology — Eurocrypt 2010*, volume 6110 of *LNCS*, pages 157–176. Springer, 2010.
- [57] Adam Groce and Jonathan Katz. Fair computation with rational players. In *Eurocrypt*, pages 81–98, 2012.
- [58] Adam Groce, Jonathan Katz, Aishwarya Thiruvengadam, and Vassilis Zikas. Byzantine agreement with a rational adversary. In *ICALP*, pages 561–572, 2012.
- [59] Iftach Haitner and Eran Omri. Coin flipping with constant bias implies one-way functions. In *FOCS*, pages 110–119, 2011.
- [60] Joseph Y. Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: Extended abstract. In László Babai, editor, *36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 623–632. ACM Press, June 2004.
- [61] Y. Huang, J. Katz, and D. Evans. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
- [62] Pavel Hubacek, Jesper Nielsen, and Alon Rosen. Limits on the power of cryptographic cheap talk. In *Crypto (1)*, pages 277–297, 2013.
- [63] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In *9th Theory of Cryptography Conference — TCC 2012*, LNCS, pages 21–38. Springer, 2012.
- [64] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.
- [65] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous communication. In *TCC*, pages 477–498, 2013.
- [66] Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
- [67] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.
- [68] Alptekin Küpcü and Anna Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS 2010: 15th European Symposium on Research in Computer Security (ESORICS)*, LNCS, pages 488–507. Springer, 2010.
- [69] Alptekin Küpcü and Anna Lysyanskaya. Usable optimistic fair exchange. In *Cryptographers' Track — RSA 2010*, LNCS, pages 252–267. Springer, 2010.
- [70] Eyal Kushilevitz and Tal Rabin. Fair e-lotteries and e-casinos. In David Naccache, editor, *Cryptographers' Track — RSA 2001*, volume 2020 of *LNCS*, pages 100–109. Springer, April 2001.
- [71] Matt Lepinski, Silvio Micali, Chris Peikert, and Abhi Shelat. Completely fair sfe and coalition-safe cheap talk. In *23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–10. ACM Press, July 2004.
- [72] Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkitasubramaniam. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In *41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 179–188. ACM Press, 2009.
- [73] Andrew Y. Lindell. Legally-enforceable fairness in secure two-party computation. In Tal Malkin, editor, *Cryptographers' Track — RSA 2008*, volume 4964 of *LNCS*, pages 121–137. Springer, April 2008.
- [74] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Crypto (2)*, pages 1–17.

- [75] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and adversarial behavior in multi-party computation (extended abstract). In Cynthia Dwork, editor, *Advances in Cryptology — Crypto 2006*, volume 4117 of *LNCS*, pages 180–197. Springer, 2006.
- [76] Hemanta K. Maji, Manoj Prabhakaran, and Amit Sahai. On the computational complexity of coin flipping. In *51st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 613–622. IEEE, 2010.
- [77] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay: a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM’04, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.
- [78] G. Maxwell. Zero knowledge contingent payment. [https://en.bitcoin.it/wiki/Zero\\_Knowledge\\_Contingent\\_Payment](https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment), 2011.
- [79] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 12–19. ACM Press, July 2003.
- [80] Silvio Micali, Rafael Pass, and Alon Rosen. Input-indistinguishable computation. In *47th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 367–378. IEEE, October 2006.
- [81] I. Miers, C. Garman, M. Green, and A. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Security and Privacy*, pages 397–411, 2013.
- [82] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for long-term data preservation. In *IEEE Security and Privacy*, 2014.
- [83] Tal Moran, Moni Naor, and Gil Segev. An optimally fair coin toss. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 1–18. Springer, 2009.
- [84] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. <http://bitcoin.org/bitcoin.pdf>.
- [85] Moni Naor. Bit commitment using pseudo-randomness. In Gilles Brassard, editor, *Advances in Cryptology — Crypto ’89*, volume 435 of *LNCS*, pages 128–136. Springer, 1990.
- [86] A. Narayanan, J. Bonneau, A. Miller, J. Clark, and J. Kroll. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography*, 2014.
- [87] J. Nielsen, P. Nordholt, C. Orlandi, and S. Burra. A new approach to practical active-secure two-party computation. In *Crypto*, 2012.
- [88] Shien Jin Ong, David C. Parkes, Alon Rosen, and Salil P. Vadhan. Fairness with an honest minority and a rational majority. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 36–53. Springer, 2009.
- [89] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — Crypto ’91*, volume 576 of *LNCS*, pages 129–140. Springer, 1992.
- [90] Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *Advances in Cryptology — Eurocrypt 2003*, volume 2656 of *LNCS*, pages 87–105. Springer, 2003.
- [91] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. In *Commun. ACM* 43(12), pages 45–48.
- [92] Meni Rosenfeld. Analysis of hashrate-based double-spending. <http://arxiv.org/abs/1402.2009>, 2012.
- [93] C-H. Wang. Efficient and practical fair exchange with off-line ttp. In *IEEE Security and Privacy*, pages 77–85, 1998.
- [94] C-H. Wang. Untraceable fair network payment protocols with off-line ttp. In *Asiacrypt*, pages 173–187, 2003.
- [95] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

## A Missing Ideal Functionalities

The (unfair) ideal functionality  $\mathcal{F}_f$  guaranteeing security with agreement on abort is presented in Figure 4. The special ideal functionality  $\mathcal{F}_{\text{rec}}^*$  guaranteeing fair reconstruction (with penalties) is presented in Figure 5.

## B Secure Computation with Penalties—More Details

In Figure 6, we present a pictorial representation of our Roof and Ladder deposit phases for fair reconstruction. We present the protocol for  $\mathcal{F}_{\text{rec}}^*$  in the  $\mathcal{F}_{\text{CR}}^*$ -hybrid model in Figure 7.

Functionality  $\mathcal{F}_f$

$\mathcal{F}_f$  with session identifier  $sid$  proceeds as follows, running with parties  $P_1, \dots, P_n$ , a parameter  $1^\lambda$ , and an adversary  $\mathcal{S}$  that corrupts parties  $\{P_s\}_{s \in C}$ :

- *Input phase:* Wait to receive a message  $(\text{input}, sid, ssid, r, y_r)$  from  $P_r$  for all  $r \in H$ . Then wait to receive a message  $(\text{input}, sid, ssid, s, \{y_s\}_{s \in C})$  from  $\mathcal{S}$ .
- *Output delivery:*
  - Compute  $(z_1, \dots, z_n) \leftarrow f(y_1, \dots, y_n)$ , and send message  $(\text{output}, sid, ssid, \{z_s\}_{s \in C})$  to  $\mathcal{S}$ .
  - If  $\mathcal{S}$  returns  $(\text{continue}, sid, ssid)$ , then send  $(\text{output}, sid, ssid, z_r)$  to  $P_r$  for all  $r \in H$ .
  - Else if  $\mathcal{S}$  returns  $(\text{abort}, sid, ssid)$ , then send  $(\text{output}, sid, ssid, \perp)$  to  $P_r$  for all  $r \in H$ .

Figure 4: The (unfair) ideal functionality  $\mathcal{F}_f$  guaranteeing security with agreement on abort.

Functionality  $\mathcal{F}_{\text{rec}}^*$

$\mathcal{F}_{\text{rec}}^*$  with session identifier  $sid$  proceeds as follows, running with parties  $P_1, \dots, P_n$ , a parameter  $1^\lambda$ , and an adversary  $\mathcal{S}$  that corrupts parties  $\{P_s\}_{s \in C}$ : Let  $H = [n] \setminus C$  and  $h = |H|$ . Let  $d$  denote the safety deposit, and let  $q$  denote the penalty amount. In addition, we assume  $\mathcal{F}_{\text{rec}}^*$  is parameterized by the pubNMSS scheme specified in Section 3.

- *Input phase:* Wait to receive a message  $(\text{input}, sid, ssid, r, \text{AllTags}, \text{Token}_r, \text{coins}(d))$  from  $P_r$  for all  $r \in H$ . Then wait to receive a message  $(\text{input}, sid, ssid, \{\text{Token}_s\}_{s \in C}, H', \text{coins}(h'q))$  from  $\mathcal{S}$  where  $h' = |H'|$ .
- *Output phase:*
  - Send  $(\text{return}, sid, ssid, \text{coins}(d))$  to each  $P_r$  for  $r \in H$ .
  - Compute  $z \leftarrow \text{Rec}(\text{AllTags}, \text{Token}_1, \dots, \text{Token}_n)$ , where  $\text{Rec}$  is the reconstruction function for pubNMSS.
    - If  $h' = 0$ , then send message  $(\text{output}, sid, ssid, z)$  to  $P_r$  for  $r \in [n]$ , and terminate.
    - If  $0 < h' < h$ , then send  $(\text{extra}, sid, ssid, \text{coins}(q))$  to  $P_r$  for each  $r \in H'$ , and terminate.
    - If  $h' = h$ , then send message  $(\text{output}, sid, ssid, z)$  to  $\mathcal{S}$ .
  - If  $\mathcal{S}$  returns  $(\text{continue}, sid, ssid, H'')$ , then send  $(\text{output}, sid, ssid, z)$  to  $P_r$  for all  $r \in H$ , and send  $(\text{payback}, sid, ssid, \text{coins}((h - h'')q))$  to  $\mathcal{S}$  where  $h'' = |H''|$ , and send  $(\text{extrapay}, sid, ssid, \text{coins}(q))$  to  $P_r$  for each  $r \in H''$ .
  - Else if  $\mathcal{S}$  returns  $(\text{abort}, sid, ssid)$ , send  $(\text{penalty}, sid, ssid, \text{coins}(q))$  to  $P_r$  for all  $r \in H$ .

Figure 5: The ideal functionality  $\mathcal{F}_{\text{rec}}^*$ .

**Roof DEPOSITS.**

$$\begin{array}{ccc}
 P_1 & \xrightarrow[q, \tau_n]{T_1 \wedge \cdots \wedge T_n} & P_n \\
 P_2 & \xrightarrow[q, \tau_n]{T_1 \wedge \cdots \wedge T_n} & P_n \\
 & \vdots & \\
 P_{n-2} & \xrightarrow[q, \tau_n]{T_1 \wedge \cdots \wedge T_n} & P_n \\
 P_{n-1} & \xrightarrow[q, \tau_n]{T_1 \wedge \cdots \wedge T_n} & P_n
 \end{array}$$

**Ladder DEPOSITS.**

$$\begin{array}{ccc}
 P_n & \xrightarrow[(n-1)q, \tau_{n-1}]{T_1 \wedge \cdots \wedge T_{n-1}} & P_{n-1} \\
 P_{n-1} & \xrightarrow[(n-2)q, \tau_{n-2}]{T_1 \wedge \cdots \wedge T_{n-2}} & P_{n-2} \\
 & \vdots & \\
 P_3 & \xrightarrow[2q, \tau_2]{T_1 \wedge T_2} & P_2 \\
 P_2 & \xrightarrow[q, \tau_1]{T_1} & P_1
 \end{array}$$

Figure 6: Roof and Ladder deposit phases for fair reconstruction.

**Preliminaries.** Let  $\text{AllTags} = (\text{Tag}_1, \dots, \text{Tag}_n)$  be publicly known tags such that only party  $P_s$  is given the token  $\text{Token}_s$  that corresponds to  $\text{Tag}_s$ . The tags and tokens correspond to shares obtained from pubNMSS scheme specified in Section 3. Let  $\phi(\text{Tag}, \cdot)$  be the circuit that accepts input  $T$  and outputs 1 iff  $T$  is a token that matches  $\text{Tag}$ . For  $s \in [n]$ , let  $\phi_{\text{lad}}^s = \phi(\text{Tag}_1, \cdot) \wedge \dots \wedge \phi(\text{Tag}_s, \cdot)$ . Let  $\phi_{\text{rf}} = \phi_{\text{lad}}^n$ .

#### Protocol.

1. (Roof DEPOSITS.) For  $s \in \{1, \dots, n-1\}$ , party  $P_s$  does the following:
  - Send  $(\text{deposit}, sid, ssid, s, n, \phi_{\text{rf}}, \tau_n, Q_{\text{rf}}^s = \text{coins}(q))$  to  $\mathcal{F}_{\text{CR}}^*$ .
  - If there exists  $r \in [n-1]$  such that the message  $(\text{deposit}, sid, ssid, r, n, \phi_{\text{rf}}, \tau_n, q)$  was not received from  $\mathcal{F}_{\text{CR}}^*$ , then output  $\perp$ , and if  $s \neq n$ , then wait to receive message  $(\text{refund}, sid, ssid, s, n, \phi_{\text{rf}}, \tau_n, Q_{\text{rf}}^s)$  from  $\mathcal{F}_{\text{CR}}^*$ . Terminate the protocol.
2. (Ladder DEPOSITS.) For  $s = n-1$  to 1, each  $P_{s+1}$  sends  $(\text{deposit}, sid, ssid, s+1, s, \phi_{\text{lad}}^s, \tau_s, Q_{\text{lad}}^{s+1} = \text{coins}(s \cdot q))$  to  $\mathcal{F}_{\text{CR}}^*$  only if for each  $r = n-1$  to  $s+1$ , the message  $(\text{deposit}, sid, ssid, r+1, r, \phi_{\text{lad}}^r, \tau_r, r \cdot q)$  was received from  $\mathcal{F}_{\text{CR}}^*$ .
3. (Ladder CLAIMS.) For each  $s \in \{0, \dots, n-1\}$ , party  $P_{s+1}$  does the following:
  - Wait until round  $\tau_s$  to receive a message  $(\text{claim}, sid, ssid, s+1, s, \phi_{\text{lad}}^s, \tau_s, s \cdot q, W_s)$  from  $\mathcal{F}_{\text{CR}}^*$ . If such a message was received by round  $\tau_s$ , create  $W_{s+1} \leftarrow W_s \cup \{\text{Token}_{s+1}\}$ , and if  $s+1 \neq n$ , send message  $(\text{claim}, sid, ssid, s+2, s+1, \phi_{\text{lad}}^{s+1}, \tau_{s+1}, (s+1) \cdot q, W_{s+1})$  to  $\mathcal{F}_{\text{CR}}^*$ , and receive back  $(\text{claim}, sid, ssid, s+2, s+1, \phi_{\text{lad}}^{s+1}, \tau_{s+1}, Q_{\text{lad}}^{s+2})$  from  $\mathcal{F}_{\text{CR}}^*$ .
  - If no such message was received output  $\perp$ , and do:
    - Wait until round  $\tau_s + 1$  to receive message  $(\text{refund}, sid, ssid, s+1, s, \phi_{\text{lad}}^s, \tau_s, Q_{\text{lad}}^{s+1})$  from  $\mathcal{F}_{\text{CR}}^*$ .
    - If  $s+1 \neq n$ , wait until round  $\tau_n + 1$  to receive message  $(\text{refund}, sid, ssid, s+1, n, \phi_{\text{lad}}^{s+1}, \tau_n, Q_{\text{rf}}^{s+1})$  from  $\mathcal{F}_{\text{CR}}^*$ , and terminate the protocol.
4. (Roof CLAIMS.) After round  $\tau_{n-1}$ , for each  $s \in [n-1]$ , party  $P_n$  sends  $(\text{claim}, sid, ssid, s, n, \phi_{\text{rf}}, \tau_n, q, W_n)$  to  $\mathcal{F}_{\text{CR}}^*$ , following which it waits to receive message  $(\text{claim}, sid, ssid, s, n, \phi_{\text{rf}}, \tau_n, Q_{\text{rf}}^s)$ . Finally,  $P_n$  outputs  $W_n = \{\text{Token}_1, \dots, \text{Token}_n\}$ , reconstructs the secret, and terminates the protocol.
5. (Token COLLECTION.) For each  $s \in \{1, \dots, n-1\}$ , party  $P_s$  waits until round  $\tau_n$  to receive a message  $(\text{claim}, sid, ssid, s', n, \phi_{\text{rf}}, \tau_n, q, W_n)$  from  $\mathcal{F}_{\text{CR}}^*$  for some  $s' \in [n-1]$ . If such a message is received for some  $s' \in [n-1]$ , then use  $W_n = \{\text{Token}_1, \dots, \text{Token}_n\}$  to reconstruct the secret. If no such message is received for  $s' = s$ , then  $P_s$  waits until time  $\tau_n + 1$  to receive message  $(\text{refund}, sid, ssid, s, n, \phi_{\text{rf}}, \tau_n, Q_{\text{rf}}^s)$  from  $\mathcal{F}_{\text{CR}}^*$ .

Figure 7: Realizing  $\mathcal{F}_{\text{rec}}^*$  in the  $\mathcal{F}_{\text{CR}}^*$ -hybrid model.

## C Related Work

**Fairness in standard secure computation.** Fair two party coin tossing was shown to be impossible in [35]. Fair secure computation for restricted classes of functions was shown in [53, 5]. Protocols for partially fair secure computation were constructed and improved in [56, 83, 15, 14, 63]. Several works have discussed the relations between coin tossing and one-way functions. See [37, 36, 59, 76] and references therein. A comprehensive study of complete primitives for fairness can be found in [54].

**Gradual release mechanisms.** Starting from early works [13, 50], gradual release mechanism have been employed to solve the problem of fair exchange in several situations [22, 42, 46]. A good survey of this area can be found in [90]. A formal treatment of gradual release mechanisms can be found in [45].

**Optimistic model.** The earliest works on optimistic fair exchange were [9, 8, 18]. Since then, there has been a huge body of work that deals with optimistic models for fair exchange (e.g., [69, 68, 79, 43, 93, 10] and references therein). Optimistic models for secure computation was considered in [23]. [69] consider a model similar to ours where receiving payment in the event of breach of fairness is also considered fair. The work of [68] shows some negative results indicating that optimistic fair exchange protocols in settings with multiple arbiters requires timeouts. [38] design non-interactive protocols for optimistic fair exchange. Optimistic fair exchange in the multiuser setting was studied in [40], and constructions using one-way functions in the random oracle model, and trapdoor one-way permutations in the standard model were shown.

**Incentivized computation and reputation systems.** Golle and Mironov [52] consider a setting where a supervisor incentivizes users to deter cheating attempts in computations involving hard functions. Assuming a trusted arbiter, Belenkiy *et al.* [17] design a credit system where users are rewarded for good work and fined for cheating, and also employ bounty mechanisms to catch cheaters. See [52, 17] and references therein for more discussion.

Reputation systems have been constructed and analyzed in several papers (see, e.g. [91] for an overview). Fair secure computation with reputation systems was considered in [6]. It has been claimed that reputations systems find limited applicability because it is unclear how to define the reputation of new users (especially since users can pick new names whenever they want)[41].

**Legally enforceable fairness.** Chen, Kudla, and Paterson [33] designed protocols for fair exchange of signatures in a model where signatures are validated only in a court-of-law. Following this, Lindell [73] showed how to construct legally enforceable fairness in the two party secure computation where parties have access to a trusted bank (or a court of law).

**E-cash and cryptographic currency.** Starting from Chaum’s work [31], there has been a lot of work proposing use of e-cash for protecting privacy of financial transactions. (See [94, 68, 17, 24, 25] and references therein.) In contrast with traditional e-cash that requires a central trusted bank, recent years have seen the emergence of new decentralized systems that allow anonymous spending of coins. Today cryptographic currencies like Bitcoin [84] and Litecoin provide various practical alternatives to traditional currency. There are several proposed forks/extensions of Bitcoin such as Mixcoin [86], Permacoin [82], Zerocash [81, 20], etc.

**Rational adversaries.** There is a long line of work that attempts to unify cryptography and game theory. Applications of cryptography to game theory include the works of [39, 1, 2, 62]. More directly related to secure computation are the works of [75, 57, 58, 55, 60, 88, 71]. Protocols for multiparty lottery are also designed in [70, 4, 11].

## D Proof of Theorem 1

### D.1 Honest binding commitments

In this section, we describe honest binding commitments and present some constructions. Our presentation is taken almost verbatim from [44] where they show how to use such commitments to obtain *adaptively secure universally composable* broadcast. Note we do not require adaptive security in our constructions, yet these commitments are exactly what we need for non-malleable secret sharing with public reconstruction. Finally, recall that our constructions require non-black-box use of such commitments since the commitments need to be generated as output of a secure evaluation of the augmented function  $\hat{f}$ . We proceed to the technical details below.

**Definition 4.** A (non-interactive) *commitment scheme* for message space  $\{\mathcal{M}_\lambda\}$  is a pair of PPT algorithms  $S, R$  such that for all  $\lambda \in \mathbb{N}$ , all messages  $m \in \mathcal{M}_k$ , and all random coins  $\omega$  it holds that  $R(m, S(1^\lambda, m; \omega), \omega) = 1$ .

A commitment scheme for message space  $\{\mathcal{M}_\lambda\}$  is *honest-binding* if it satisfies the following:

**Binding (for an honest sender)** For all PPT algorithms  $\mathcal{A}$  (that maintain state throughout their execution), the following is negligible in  $\lambda$ :

$$\Pr[m \leftarrow \mathcal{A}(1^\lambda); \omega \leftarrow \{0, 1\}^*; \text{com} \leftarrow S(1^\lambda, m; \omega); (m', \omega') \leftarrow \mathcal{A}(\text{com}, \omega) : R(m', \text{com}, \omega') = 1 \wedge m' \neq m]$$

**Equivocation** There is an algorithm  $\tilde{S} = (\tilde{S}_1, \tilde{S}_2)$  such that for all PPT  $\mathcal{A}$  (that maintain state throughout their execution) the following is negligible:

$$\left| \Pr[m \leftarrow \mathcal{A}(1^\lambda); \omega \leftarrow \{0, 1\}^*; \text{com} \leftarrow S(1^\lambda, m; \omega) : \mathcal{A}(1^\lambda, \text{com}, \omega) = 1] - \Pr[(\text{com}, \text{st}) \leftarrow \tilde{S}_1(1^\lambda); m \leftarrow \mathcal{A}(1^\lambda); \omega \leftarrow \tilde{S}_2(\text{st}, m) : \mathcal{A}(1^\lambda, \text{com}, \omega) = 1] \right|$$

Equivocation implies the standard hiding property, namely, that for all PPT algorithms  $\mathcal{A}$  (that maintain state throughout their execution) the following is negligible:

$$\left| \Pr[(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda); b \leftarrow \{0, 1\}; \text{com} \leftarrow S(1^\lambda, m_b) : \mathcal{A}(\text{com}) = b] - \frac{1}{2} \right|.$$

We also observe that if  $(\text{com}, \omega)$  are generated by  $(\tilde{S}_1, \tilde{S}_2)$  for some message  $m$  as in the definition above, then binding still holds: namely, no PPT adversary can find  $(m', \omega')$  with  $m' \neq m$  such that  $R(m', \text{com}, \omega') = 1$ .

**Constructing honest-binding commitments.** We show two constructions of honest-binding commitment schemes. The proofs that these schemes satisfy Definition 4 are relatively straightforward.

The first construction, based on the commitment scheme of Naor [85], relies on the minimal assumption that one-way functions exist. We describe the scheme for committing single-bit messages, though it could be extended to arbitrary length messages in the obvious way. In the following,  $G$  is a length-tripling pseudorandom generator.

$$\begin{array}{c}
\frac{S(1^k, m; \omega)}{\text{parse } \omega \text{ as } \text{crs} \| r, \\
\text{with } |\text{crs}| = 3\lambda \\
\text{and } |r| = \lambda; \\
c := G(r) \oplus (\text{crs} \cdot m); \\
\text{com} := (\text{crs}, c); \\
\text{return com};}
\end{array}
\qquad
\begin{array}{c}
\frac{R(m, (\text{crs}, c), \omega)}{\text{parse } \omega \text{ as } \text{crs} \| r, \\
\text{with } |\text{crs}| = 3\lambda \\
\text{and } |r| = \lambda; \\
\text{if } c \stackrel{?}{=} G(r) \oplus (\text{crs} \cdot m) \\
\quad \text{return 1;} \\
\text{else return 0};}
\end{array}$$

$$\begin{array}{c}
\frac{\tilde{S}_1(1^k)}{r_0, r_1 \leftarrow \{0, 1\}^\lambda; \\
\text{crs} := G(r_0) \oplus G(r_1); \\
c := G(r_0); \\
\text{com} := (\text{crs}, c); \\
\text{st} := (r_0, r_1, \text{com}); \\
\text{return } (\text{com}, \text{st});}
\end{array}
\qquad
\begin{array}{c}
\frac{\tilde{S}_2(\text{st}, m)}{\text{parse st as } (r_0, r_1, \text{com}); \\
\text{parse com as } (\text{crs}, c); \\
\text{if } m \stackrel{?}{=} 0 \\
\quad \omega := \text{crs} \| r_0; \\
\text{else} \\
\quad \omega := \text{crs} \| r_1; \\
\text{return } \omega;}
\end{array}$$

Next, we show an efficient scheme that allows for direct commitments to strings. This construction, based on the Pedersen commitment scheme [89], relies on the discrete-logarithm assumption. In the following, we let  $\mathbb{G}$  be a cyclic group of order  $q$ , with generator  $g \in \mathbb{G}$ .

$$\begin{array}{c}
\frac{S(1^k, m; \omega)}{\text{Parse } \omega \text{ as } h \| x, \\
\text{with } h \in \mathbb{G} \\
\text{and } x \in \mathbb{Z}_q; \\
\text{return com} := (h, g^m h^x);}
\end{array}
\qquad
\begin{array}{c}
\frac{R(m, \text{com}, \omega)}{\text{Parse } \omega \text{ as } h \| x, \\
\text{with } h \in \mathbb{G} \\
\text{and } x \in \mathbb{Z}_q; \\
\text{if com} \stackrel{?}{=} (h, g^m h^x) \\
\quad \text{return 1;} \\
\text{else return 0};}
\end{array}$$
  

$$\begin{array}{c}
\frac{\tilde{S}_1(1^\lambda)}{r, y \leftarrow \mathbb{Z}_q; \\
\text{com} := (g^r, g^y); \\
\text{return } (\text{com}, (r, y))}
\end{array}
\qquad
\begin{array}{c}
\frac{\tilde{S}_2((r, y), m)}{\text{if } r \stackrel{?}{=} 0 \text{ return } \perp; \\
x := (y - m) \cdot r^{-1} \bmod q; \\
\text{return } \omega := g^r \| x;}
\end{array}$$

For a even more practical instantiation one may construct heuristically secure honest binding commitment schemes in the *programmable random oracle* model. In the following let  $\text{Hash}$  be a programmable hash function, and let  $\omega \in \{0, 1\}^\lambda$ . We only describe the algorithms  $S, R$ . The algorithms  $\tilde{S}_1, \tilde{S}_2$  are obtained by standard programming techniques.

$$\begin{array}{c}
\frac{S(1^k, m; \omega)}{\text{return com} := \text{Hash}(m \| \omega);}
\end{array}
\qquad
\begin{array}{c}
\frac{R(m, \text{com}, \omega)}{\text{If com} \stackrel{?}{=} \text{Hash}(m \| \omega) \\
\quad \text{return 1;} \\
\text{else return 0};}
\end{array}$$

## D.2 Proof of Security

We start with an informal claim that should provide some intuition regarding the correctness/security of our protocol.

**Proposition 2.** *For all  $s \in [n]$ , if  $P_s$  is honest, then at the end of the protocol (i.e., after round  $\tau_n$ ) either*

- both  $(n - s)$ -th ladder deposit, and (for  $s > 1$ ) the  $(n - s - 1)$ -th ladder deposit were claimed; or
- for all  $j \in [n]$ , the  $j$ -th roof deposit cannot be claimed, and for all  $j \leq n - s$ , the  $j$ -th ladder deposit cannot be claimed.

*Proof.* Note that an honest  $P_s$  does not reveal  $T_s = \text{Token}_s$  unless the  $(n - s - 1)$ -th ladder deposit is claimed. Now, there are two cases. In the first case,  $P_{s-1}$  claimed the  $(n - s - 1)$ -th ladder deposit. In this case, observe that the only extra token required by  $P_s$  to claim the  $(n - s)$ -th ladder deposit is its own token  $T_s$ . Therefore, an honest  $P_s$  would claim the  $(n - s)$ -th ladder deposit as well. In the second case,  $P_{s-1}$  did not claim the  $(n - s - 1)$  ladder deposit. In this case,  $P_s$  does not claim the  $(n - s)$ -th ladder deposit, and therefore, no (corrupt) party obtains  $T_s$ . The final observation is that unless token  $T_s$  is obtained, (1) none of the roof deposits can be claimed, and (2) for all  $j \leq n - s$ , the  $j$ -th ladder deposit cannot be claimed. The claim follows.  $\square$

The above claim implies that either (1) all honest parties received all tokens without losing any coins, or (2) each honest party that did not reveal its tokens can get all its deposits refunded while no corrupt party obtains all tokens, or (3) each honest party that reveals its own tokens but does not obtain all tokens is compensated by  $\text{coins}(s \cdot q) - \text{coins}((s - 1) \cdot q) = \text{coins}(q)$  at the end of the protocol. These are precisely the properties we required. We now proceed to the simulation.

**Overview of the simulation.** We first sketch the simulation without specifying how the simulator  $\mathcal{S}$  deals with coins. First,  $\mathcal{S}$  acts as the augmented functionality (alternatively, uses the simulator for  $\hat{f}$ ) to first extract the actual inputs of the corrupt parties (denoted  $\{y_s\}_{s \in C}$ ) and then sets the outputs of  $\mathcal{F}_{\hat{f}}$  to honestly generated tags (on random shares) for corrupt parties and equivocal tags for the honest parties. Then, as parties reveal the corresponding tokens in the ladder claim phase, the simulator decides on how to equivocate the tags. Specifically, there are two cases to handle depending on whether  $P_n$  is honest or corrupt.

- If  $P_n$  is honest, then the adversary’s decision to abort is essentially independent of the final output (of course, assume that the underlying commitments are hiding). In this case if the adversary does abort, then  $\mathcal{S}$  does not contact the ideal functionality  $\mathcal{F}_f^*$ . This simply corresponds to a “premature” abort and hence is dealt in the same way as in standard secure computation. On the other hand if the adversary does not abort, then  $\mathcal{S}$  provides the extracted inputs to  $\mathcal{F}_f^*$  and receives back the actual outputs. It then equivocates the last honest party’s (equivocal) tags using the equivocal simulator  $\tilde{S}_2$  (specified in Section D.1), such that the revealed tokens/shares now correspond to an additive sharing of the final output.
- Else if  $P_n$  is dishonest, then the adversary’s decision to abort may depend on the value of the final output. Of course, once again, the adversary may abort prematurely, and this is dealt exactly as in the previous case. If the adversary does not abort prematurely, then  $\mathcal{S}$  will contact the ideal functionality  $\mathcal{F}_f^*$  when it has to reveal the tokens for the last honest party. Specifically, the simulator will send the extracted inputs to  $\mathcal{F}_f^*$ , receive back the outputs, and then equivocate the last honest party’s tag using the equivocal simulator  $\tilde{S}_2$  such that the revealed tokens/shares correspond to an additive sharing of the final output.

The tricky part in the simulation is to precisely define how the simulator deals with coins. Specifically, we need to show that the simulator is able to provide indistinguishable views (where view includes distribution of coins). The problem is made more difficult by the restriction that only  $\mathcal{Z}$  (i.e., not  $\mathcal{S}$ ) can “create” (or destroy) coins. Therefore,  $\mathcal{S}$  must succeed in the above only by using coins it received from  $\mathcal{A}$ . In our simulation, we define wallets  $\text{SimWallet}_j$  for each  $j \in C$ , and  $\text{PenaltyWallet}$  to better capture the “flow” of coins provided by the adversary. Specifically,  $\text{SimWallet}_j$  accounts for coins provided by  $P_j$  (controlled by  $\mathcal{A}$ ). On the other hand,  $\text{PenaltyWallet}$  accounts for coins (transferred to/from  $\{\text{SimWallet}_j\}_{j \in C}$ ) that  $\mathcal{S}$  sends to the ideal functionality. (Throughout our simulation, we make an effort to specify the state of these wallets after every stage of the protocol.)

We now provide a very high level overview of some concrete difficulties that we need to address in the simulation, and how we handle these. It is relatively easy to handle premature aborts by  $\mathcal{A}$  during the roof deposits and ladder deposits. Simulating the ladder claims is perhaps the most involved part of the simulation.

This is in part amplified by the fact that each corrupt party does not necessarily have to reveal its individual token, but this token may eventually be revealed by a corrupt party (immediately) above it in the ladder. Another important issue to address is the specification of the set  $H'$  which  $\mathcal{S}$  is required to send to  $\mathcal{F}_f^*$ , and further on how to assign  $|H'|q$  coins to `PenaltyWallet`. (We note that specifying  $H''$  is relatively easy and essentially corresponds to the case when the last party is corrupt, and chooses to claim some, but not all, of the roof deposits.) We specify  $H'$  as the set of honest parties that revealed their tokens in the ladder claim phase of the protocol. *A priori* it is not even clear how  $\mathcal{S}$  can afford to assign  $|H'|q$  coins to `PenaltyWallet` while ensuring that corrupt  $P_j$  receives coins( $jq$ ) when it correctly makes its ladder claim. We do this in the following way. When a honest party reveals its tokens, we add the index of this party to  $H'$  and move coins( $q$ ) from  $\text{SimWallet}_{j'}$  to `PenaltyWallet` where  $P_{j'}$  is the corrupt party that is immediately above this honest party in the ladder. Similarly, when a corrupt party  $P_j$  claims coins( $jq$ ) by revealing its token,  $\mathcal{S}$  sends coins( $jq$ ) to  $\mathcal{A}$  where coins( $jq$ ) are moved from  $\text{SimWallet}_{j'}$  where  $P_{j'}$  is the corrupt party that is immediately above  $P_j$  in the ladder. This is essentially the main idea behind our simulation. In this overview, we omit further discussion of boundary conditions. These are fully specified in the formal simulation described below.

*Proof.* Here we sketch the simulation. To keep the proof simple, we assume that all parties receive the same output. Our proof can be extended in a straightforward way to accommodate the case when different parties receive different outputs. A main issue to address would be to prove that the simulator does not need extra “resources” (i.e., coins) to do the simulation. We show that the simulation is *straightline* both in how it deals with coins as well as on how it deals with tags and tokens.

*Simulating secure evaluation of the augmented function.* In the first phase, the simulator acts as the augmented functionality  $\mathcal{F}_{\hat{f}}$  and performs the following:

- The simulator extracts the inputs of all corrupt parties by acting as  $\mathcal{F}_{\hat{f}}$ . Let the inputs be  $\{y_j\}_{j \in C}$ .
- For all  $j \in C$ , run the honest tag generation algorithm with random  $\text{sh}_j$  and uniform randomness to create  $(\text{Tag}'_j, \text{Token}'_j)$ .
- For all  $i \in H$ , run the simulated tag generation algorithm to create  $\text{Tag}'_i = \text{SimTag}_i$  and save state  $\text{st}_i$ .
- Set  $\text{AllTags}' = \{\text{Tag}'_j\}_{j \in [n]}$ , and send  $\text{AllTags}', \{(\text{Tag}'_j, \text{Token}'_j)\}_{j \in C}$  to  $\mathcal{A}$  controlling  $\{P_j\}_{j \in C}$ .

The simulator also initializes  $H' = \emptyset$ . The simulator also keeps track of coins received from malicious party  $P_j$  in  $\text{SimWallet}_j$ . (Obviously,  $\text{SimWallet}_j$  does not contain any coins at this stage.)

*Simulating roof deposits.* In the roof deposit phase, the simulator acting as  $\mathcal{F}_{\text{CR}}^*$  first initializes  $\text{rfdep} = 1$  ( $\text{rfdep}$  indicates whether the roof deposit phase was executed without any aborts), and  $\text{rfd}_j = 0$  for all  $j \in C \setminus \{n\}$  ( $\text{rfd}_j$  indicates whether  $P_j$  made its roof deposit). Note in the following  $\phi_{\text{rf}}$  is determined by the tags generated by the simulator as above. In the roof deposit phase, the simulator performs the following:

- For every  $i \in H \setminus \{n\}$ , acting as  $\mathcal{F}_{\text{CR}}^*$  the simulator sends message  $(\text{deposit}, \text{sid}, \text{ssid}, i, n, \phi_{\text{rf}}, \tau_n, q)$  to  $\mathcal{A}$ .
- For every  $j \in C \setminus \{n\}$ , the simulator waits to receive messages  $(\text{deposit}, \text{sid}, \text{ssid}, j, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  from  $\mathcal{A}$  controlling  $P_j$ .
  - If the message was received, then the simulator adds coins( $q$ ) to  $\text{SimWallet}_j$  and updates  $\text{rfd}_j = 1$ .
  - If the message was not received  $\mathcal{S}$  sets  $\text{rfdep} = 0$ .

Suppose  $\text{rfdep} = 0$ , then the simulator waits until time  $\tau_n + 1$ , and then it acts as  $\mathcal{F}_{\text{CR}}^*$  and does the following: (1) sends for every  $j \in C \setminus \{n\}$  satisfying  $\text{rfd}_j = 1$  the message  $(\text{refund}, \text{sid}, \text{ssid}, j, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  to  $\mathcal{A}$  (controlling  $P_j$ ) where coins( $q$ ) are taken from  $\text{SimWallet}_j$ , and (2) sends the message  $(\text{refund}, \text{sid}, \text{ssid}, i, n, \phi_{\text{rf}}, \tau_n, q)$  to  $\mathcal{A}$  for  $i \in H$  or  $i \in C \setminus \{n\}$  with  $\text{rfd}_i = 1$ . Note at this point that for all  $j \in C$ ,  $\text{SimWallet}_j$  contains no coins. Now,  $\mathcal{S}$  terminates the simulation, and outputs whatever  $\mathcal{A}$  outputs. Thus the adversary does not obtain any output, and in particular  $\mathcal{S}$  did not send any message to  $\mathcal{F}_f^*$  and thus, honest parties obtain no output either.

On the other hand, if  $\text{rfdep} = 1$ , then it is easy to see that for each  $j \in C \setminus \{n\}$ ,  $\text{SimWallet}_j$  contains coins( $q$ ), and  $\mathcal{S}$  continues with the simulation. Note at this stage,  $H'$  is still the empty set.

*Simulating ladder deposits.* In the ladder deposit phase, the simulator acting as  $\mathcal{F}_{\text{CR}}^*$  first initializes  $\text{laddep} = 1$  ( $\text{laddep}$  indicates whether the ladder deposit phase was executed without any aborts), and  $\text{ldd}_j = 0$  for all  $j \in C$  ( $\text{ldd}_j$  indicates if  $P_j$  made a ladder deposit or not). Let  $\hat{\tau}_k$  denote the round in which  $P_{k+1}$  is required to make its ladder deposit. (Note  $\hat{\tau}_{n-1} < \hat{\tau}_{n-2} < \dots < \hat{\tau}_1$  while  $\tau_n > \tau_{n-1} > \tau_{n-2} > \dots > \tau_1$ . Further  $\tau_1 > \hat{\tau}_1$ .) Then for each  $k = n - 1, \dots, 1$ , the simulator performs the following in round  $\hat{\tau}_k$ :

- If  $\text{laddep} = 1$  and if  $k + 1 \in H$ , then the simulator acts as  $\mathcal{F}_{\text{CR}}^*$  and sends  $(\text{deposit}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, kq)$  to  $\mathcal{A}$ .
- If  $k + 1 \in C$ , then the simulator waits to receive message  $(\text{deposit}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  from  $\mathcal{A}$  controlling  $P_{k+1}$ .
  - If the message was received, then the simulator adds  $\text{coins}(kq)$  to  $\text{SimWallet}_{k+1}$  and updates  $\text{ldd}_{k+1} = 1$ .
  - If the message was not received and if there exists  $k' < k + 1$  such that  $k' \in H$ , then  $\mathcal{S}$  sets  $\text{laddep} = 0$ .

In the following, let  $k' \in H$  be such that for all  $k < k'$  it holds that  $k \in C$ . Suppose  $\text{laddep} = 0$ , then intuitively the adversary does not learn the final output in the real execution unless it breaks the hiding property of the underlying commitment corresponding to  $\text{Tag}'_{k'}$  (whose decommitment is held only by honest  $P_{k'}$  from above). In particular, the relation  $\phi_{\text{lad}}^{k''}$  for each  $k'' \geq k'$  is designed such that the  $k''$ -th ladder deposits cannot be claimed without a valid  $\text{Token}'_{k'}$  corresponding to  $\text{Tag}'_{k'}$ . Thus when  $\text{laddep} = 0$ , the adversary does not learn any output, and in particular simulator will not send any message to  $\mathcal{F}_f^*$  and thus, honest parties obtain no output. In fact, the simulator only needs to sends refund messages in the ladder claim phase (for  $k > k'$ ) and the roof claim phase. Formally, the simulation proceeds as follows when  $\text{laddep} = 0$ :

- For  $k \in [n - 1]$ : in round  $\tau_k$  (i.e., “ladder claim” phase), do:
  - If  $\mathcal{A}$  acting as  $P_k$  for  $k \in C$  and  $k > k'$ , sends a message  $(\text{claim}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, kq, W_k)$  such that  $W_k = \{\text{Token}'_1, \dots, \text{Token}'_k\}$  such that for each  $j \in [k]$ , it holds that  $\text{Token}'_j$  is a valid token corresponding  $\text{Tag}'_j$ , then  $\mathcal{S}$  outputs  $\text{fail}_{\text{hide}}$  and aborts the simulation.
  - If  $k + 1 \in C$  and  $k + 1 < k'$ , then  $\mathcal{S}$  first checks if  $\text{ldd}_{k+1} = 1$ . If  $\text{ldd}_{k+1} = 1$ , then:
    - If  $\mathcal{A}$  acting as  $P_k$  for  $k \in C$  and  $k < k'$ , sends a message  $(\text{claim}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, kq, W_k)$  such that  $W_k = \{\text{Token}'_1, \dots, \text{Token}'_k\}$  such that for each  $j \in [k]$ , it holds that  $\text{Token}'_j$  is a valid token corresponding  $\text{Tag}'_j$ , then  $\mathcal{S}$  sends  $(\text{claim}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  to  $\mathcal{A}$  controlling  $P_k$  where  $\text{coins}(kq)$  are taken from  $\text{SimWallet}_{k+1}$ . (Note at this point if  $k + 1 \neq n$ , then  $\text{SimWallet}_{k+1}$  contains exactly  $\text{coins}(q)$  corresponding to  $P_{k+1}$ 's roof deposit. On the other hand if  $k + 1 = n$ , then  $\text{SimWallet}_{k+1}$  does not contain any coins.) Finally,  $\mathcal{S}$  sends  $(\text{claim}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, kq, W_k)$  to  $\mathcal{A}$ .

and in round  $\tau_k + 1$ , do:

- If  $k + 1 \in C$  and  $k + 1 < k'$ , then  $\mathcal{S}$  first checks if  $\text{ldd}_{k+1} = 1$ . If  $\text{ldd}_{k+1} = 1$ , then
  - If  $k \in H$ , then  $\mathcal{S}$  sends  $(\text{refund}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  to  $\mathcal{A}$  controlling  $P_{k+1}$  where  $\text{coins}(kq)$  are taken from  $\text{SimWallet}_{k+1}$ . (Note at this point if  $k + 1 \neq n$ , then  $\text{SimWallet}_{k+1}$  contains exactly  $\text{coins}(q)$  corresponding to  $P_{k+1}$ 's roof deposit. On the other hand if  $k + 1 = n$ , then  $\text{SimWallet}_{k+1}$  does not contain any coins.)
  - If  $k + 1 \in C$  and  $k + 1 > k'$ , then  $\mathcal{S}$  sends  $(\text{refund}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  to  $\mathcal{A}$  controlling  $P_{k+1}$  if  $\text{ldd}_{k+1} = 1$ , where  $\text{coins}(kq)$  are taken from  $\text{SimWallet}_{k+1}$ . (Note at this point if  $k + 1 \neq n$ , then  $\text{SimWallet}_{k+1}$  contains exactly  $\text{coins}(q)$  corresponding to  $P_{k+1}$ 's roof deposit. On the other hand if  $k + 1 = n$ , then  $\text{SimWallet}_{k+1}$  does not contain any coins.)
  - If  $k + 1 \in H$  and  $k + 1 > k'$ , then  $\mathcal{S}$  sends  $(\text{refund}, sid, ssid, k + 1, k, \phi_{\text{lad}}^k, \tau_k, kq)$  to  $\mathcal{A}$ .
- In round  $\tau_n$  (i.e., “roof claim” phase) if  $n \in C$ , and for any  $k \in [n - 1]$ : If  $\mathcal{A}$  acting as  $P_n$  sends a message  $(\text{claim}, sid, ssid, k, n, \phi_{\text{rf}}, \tau_n, q, W_n)$  such that  $W_n = \{\text{Token}'_1, \dots, \text{Token}'_n\}$  such that for each  $j \in [n]$ , it holds that  $\text{Token}'_j$  is a valid token corresponding  $\text{Tag}'_j$ , then  $\mathcal{S}$  outputs  $\text{fail}_{\text{hide}}$  and aborts the simulation.

- In round  $\tau_n + 1$  (i.e., “roof refund” phase), for each  $k \in [n - 1]$ ,  $\mathcal{S}$  acts as  $\mathcal{F}_{\text{CR}}^*$  and does the following:
  - Send  $(\text{refund}, sid, ssid, k, n, \phi_{\text{rf}}, \tau_n, q)$  to  $\mathcal{A}$ .
  - If  $k \in C$ , then send  $(\text{refund}, sid, ssid, k, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  to  $\mathcal{A}$  controlling  $P_k$ , where  $\text{coins}(q)$  are taken from  $\text{SimWallet}_k$ . (Note at this point  $\text{SimWallet}_k$  does not contain any coins.)

This completes the description of the simulation in the case when  $\text{laddep} = 0$ . Note in particular that  $\mathcal{S}$  returned all the coins it received from  $\mathcal{A}$  and further for each  $j \in C$ , it holds that  $\text{SimWallet}_j$  contains no coins.

We now focus on the remaining case, i.e., when  $\text{laddep} = 1$ . Let  $k' \in H$  be such that for all  $k < k'$  it holds that  $k \in C$ . Then when  $\text{laddep} = 1$ , it holds that  $\text{ldd}_j = 1$  for each  $j \in C$  such that  $j > k'$ . (Note for  $j \in C$  with  $j < k'$  the value  $\text{ldd}_j$  may or may not equal 1.) In any case, observe that for  $j \neq n$ , if  $\text{ldd}_j = 1$ , then  $\text{SimWallet}_j$  contains  $\text{coins}(jq) = \text{coins}((j-1)q) + \text{coins}(q)$ , where  $\text{coins}((j-1)q)$  was  $P_j$ 's ladder deposit made to  $P_{j-1}$ , and  $\text{coins}(q)$  was  $P_j$ 's roof deposit made to  $P_n$ . On the other hand, if  $j = n$  and  $\text{ldd}_j = 1$ , then  $\text{SimWallet}_j$  contains  $\text{coins}((n-1)q)$  (which corresponds to  $P_j$ 's ladder deposit). If  $\text{laddep} = 1$ , then  $\mathcal{S}$  continues with the simulation as described below. Note at this stage,  $H'$  is still the empty set.

Simulating ladder claims. In the following, we assume that  $\text{laddep} = 1$ . As above let  $k' \in H$  be such that for all  $k < k'$  it holds that  $k \in C$ . Then we have that  $\text{ldd}_j = 1$  for each  $j \in C$  with  $j > k'$ . Further if  $j \neq n$  and  $\text{ldd}_j = 1$ , then  $\text{SimWallet}_j$  contains  $\text{coins}(jq)$ , while if  $j = n$  and  $\text{ldd}_j = 1$ , then  $\text{SimWallet}_j$  contains  $\text{coins}((n-1)q)$ . For each  $k \in [n - 1]$ , let

- $\text{Hpred}_k$  denote  $i \in H$  such that (1)  $\text{ldd}_i = 1$  and (2) for all  $i' \in H$  with  $i' > k$  it holds that  $i' \geq i$ . If no such value exists, then we set  $\text{Hpred}_k = \infty$ . (That is,  $\text{Hpred}_k$  denotes the honest party immediately above  $P_k$  in the ladder.)
- $\text{Cpred}_k$  denote  $j \in C$  such that (1)  $\text{ldd}_j = 1$  and (2) for all  $j' \in C$  with  $j' > k$  it holds that  $j' \geq j$ . If no such value exists, then we set  $\text{Cpred}_k = \infty$ . (That is,  $\text{Cpred}_k$  denotes the corrupt party immediately above  $P_k$  in the ladder.)

(Note that for  $k \in [n - 1]$  both  $\text{Hpred}_k = \infty$  and  $\text{Cpred}_k = \infty$  cannot hold simultaneously.) Finally, set  $W_0 = \emptyset$ , and initialize  $\text{ladclm} = 1$  ( $\text{ladclm}$  represents whether the ladder claim phase is successfully completed), and initialize  $\text{ldc}_k = 0$  for all  $k \in [n - 1]$  ( $\text{ldc}_k$  represents if party  $P_k$  made its ladder claim), and  $\text{ldc}_0 = 1$ ,  $\text{ldc}_n = 0$ , and  $\text{contact} = 0$  ( $\text{contact}$  represents whether  $\mathcal{S}$  has sent a message to the ideal functionality). The simulation then proceeds as follows:

- For  $k \in [n - 1]$  such that  $\text{ldd}_{k+1} = 1$ : in round  $\tau_k$ , do:
  - If  $\text{ldc}_{k-1} = 1$  and  $k \in H$  and  $\text{Hpred}_k \neq \infty$ , then set  $\text{ldc}_k = 1$ , and choose random  $\text{sh}'_k$  and run the equivocal simulator  $\tilde{S}_2(\text{st}_k, \text{sh}'_k)$  to obtain  $\omega'_k$ . Then set  $\text{Token}'_k = (\text{sh}'_k, \omega'_k)$  and send  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq, W_k)$  to  $\mathcal{A}$  where  $W_k = W_{k-1} \cup \{\text{Token}'_k\}$ .
    - If  $\text{Cpred}_k \neq \infty$ , then add  $k$  to  $H'$  and move  $\text{coins}(q)$  from  $\text{SimWallet}_{\text{Cpred}_k}$  to  $\text{PenaltyWallet}$ .
  - If  $\text{ldc}_{k-1} = 1$  and  $k \in H$  and  $\text{Hpred}_k = \infty$  and  $\text{Cpred}_k \neq \infty$ , then
    - Send  $(\text{input}, sid, ssid, \{y_s\}_{s \in C}, H, \text{coins}(hq))$  to  $\mathcal{F}_f^*$ , where  $\text{coins}(hq)$  are taken from  $\text{PenaltyWallet}$ .
    - Receive  $(\text{output}, sid, ssid, z)$  from  $\mathcal{F}_f^*$ .
    - Choose  $\text{sh}'_k$  such that  $z \oplus \bigoplus_{j \neq k} \text{sh}'_j = \text{sh}'_k$ , and run the equivocal simulator  $\tilde{S}_2(\text{st}_k, \text{sh}'_k)$  to obtain  $\omega'_k$ . Then set  $\text{Token}'_k = (\text{sh}'_k, \omega'_k)$  and send  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq, W_k)$  to  $\mathcal{A}$  where  $W_k = W_{k-1} \cup \{\text{Token}'_k\}$ .
    - Set  $\text{contact} = 1$ .
  - If  $k \in C$ , then
    - if  $\mathcal{S}$  receives  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq, W''_k)$  from  $\mathcal{A}$  such that for each  $\text{Token}''_j \in W''_k$ , it holds that  $(\text{Tag}'_j, \text{Token}''_j)$  is a valid tag-token pair, yet  $W''_k \neq \{\text{Token}'_1, \dots, \text{Token}'_k\}$ , then  $\mathcal{S}$  outputs  $\text{fail}_{\text{bind}}$  and aborts the simulation.

- else if  $\mathcal{S}$  receives  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq, W''_k)$  from  $\mathcal{A}$  such that for each  $\text{Token}_j'' \in W''_k$ , it holds that  $(\text{Tag}'_j, \text{Token}'_j'')$  is a valid tag-token pair, yet there exists  $i \in H$  with  $i < k$  such that  $\text{ldc}_i = 0$ , then  $\mathcal{S}$  outputs  $\text{fail}_{\text{hide}}$  and aborts the simulation.
- else if  $\mathcal{S}$  receives  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq, W''_k)$  from  $\mathcal{A}$  such that for each  $\text{Token}_j'' \in W''_k$ , it holds that  $(\text{Tag}'_j, \text{Token}'_j'')$  is a valid tag-token pair, then  $\mathcal{S}$  sends  $(\text{claim}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  to  $\mathcal{A}$  controlling  $P_k$  where  $\text{coins}(kq)$  are obtained as follows:
  - If  $\text{Cpred}_k \neq \infty$ , then take  $\text{coins}(kq)$  from  $\text{SimWallet}_{\text{Cpred}_k}$ .
  - Else if  $\text{Cpred}_k = \infty$ , take  $\text{coins}(q)$  from each of  $\text{SimWallet}_j$  for all  $j \in C$  (this corresponds to  $\text{coins}(q)$  deposited by  $P_j$  in the roof deposit phase), and  $\text{coins}(h'q)$  from  $\text{PenaltyWallet}$  where  $h' = |H'|$ . Since  $\text{Cpred}_k = \infty$ , the total number of coins from above equals  $\text{coins}(tq)$  plus  $\text{coins}(h'q)$ . Note that  $t + h' = k$ , and so this way we are guaranteed to obtain  $\text{coins}(kq)$ .

In this case,  $\mathcal{S}$  updates  $\text{ldc}_k = 1$ , and sets  $W_k = W''_k$ .

- Further, if  $\text{Cpred}_k = \infty$ , then  $\mathcal{S}$  sends  $(\text{input}, sid, ssid, \{y_s\}_{s \in C}, \emptyset, \text{coins}(0))$  to  $\mathcal{F}_f^*$ , and sets  $\text{contact} = 1$ . Then,  $\mathcal{S}$  receives  $(\text{output}, sid, ssid, z)$  from  $\mathcal{F}_f^*$ . (In this case, the simulator still needs to equivocate (honest)  $P_n$ 's commitment in the roof claim phase.)
- else if  $\mathcal{S}$  did not receive any (valid) message from  $\mathcal{A}$ , then set  $\text{ldc}_k = 0$  and do:
  - if  $k+1 \in H$ , then  $\mathcal{S}$  sends  $(\text{input}, sid, ssid, \{y_s\}_{s \in C}, H', \text{coins}(h'q))$  to  $\mathcal{F}_f^*$ , where  $h' = |H'|$  and  $\text{coins}(h'q)$  are taken from  $\text{PenaltyWallet}$ . Finally, set  $\text{contact} = 1$ . (Note that in this case, at the end of the execution, it holds that  $H' \neq H$ . In particular,  $k+1 \in H$  is not contained in  $H'$ .)

and in round  $\tau_k + 1$ , do:

- If  $\text{ldc}_k = 0$  and  $k+1 \in H$ , then send message  $(\text{refund}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq)$  to  $\mathcal{A}$ .
- If  $\text{ldc}_k = 0$  and  $k+1 \in C$ , then send message  $(\text{refund}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, kq)$  to  $\mathcal{A}$  and further send message  $(\text{refund}, sid, ssid, k+1, k, \phi_{\text{lad}}^k, \tau_k, \text{coins}(kq))$  to  $\mathcal{A}$  controlling  $P_{k+1}$ , where  $\text{coins}(kq)$  are taken from  $\text{SimWallet}_{k+1}$ . (Note at this point if  $k+1 \neq n$ , then  $\text{SimWallet}_{k+1}$  contains  $\text{coins}(q)$ , else if  $k+1 = n$  then  $\text{SimWallet}_{k+1}$  contains no coins.)

*Distribution of coins.* Recall that at the beginning of the simulation of the ladder claim phase,

- if  $j \in C \setminus \{n\}$  with  $\text{ldd}_j = 1$ , then  $\text{SimWallet}_j$  contained  $\text{coins}(jq)$ .
- if  $j \in C \setminus \{n\}$  with  $\text{ldd}_j = 0$ , then  $\text{SimWallet}_j$  contained  $\text{coins}(q)$ . (Note that since  $\text{laddep} = 1$ , this implies that  $\text{rfdep} = 1$  and thus each (corrupt) party must have deposited  $\text{coins}(q)$  in the roof deposit phase.)
- if  $n \in C$  with  $\text{ldd}_n = 1$ , then  $\text{SimWallet}_n$  contained  $\text{coins}((n-1)q)$ .
- if  $n \in C$  with  $\text{ldd}_n = 0$ , then  $\text{SimWallet}_n$  contained  $\text{coins}(0)$ .
- $\text{PenaltyWallet}$  contained  $\text{coins}(0)$ .

Summarizing from the simulation above, the following are the steps in the ladder claim simulation that affect the distribution of coins for  $k = 1, \dots, n-1$ :

- If  $\text{ldc}_{k-1} = 1$  and  $k \in H$  and  $\text{Cpred}_k \neq \infty$ , then  $\text{coins}(q)$  was moved from  $\text{SimWallet}_{\text{Cpred}_k}$  to  $\text{PenaltyWallet}$ .
- If  $\text{ldc}_k = 1$  and  $k \in C$  and  $\text{Cpred}_k \neq \infty$ , then  $\text{coins}(kq)$  was moved from  $\text{SimWallet}_{\text{Cpred}_k}$  and given to  $\mathcal{A}$ .
- If  $\text{ldc}_{k-1} = 1$  and  $k \in H$  and  $\text{Hpred}_k = \infty$  and  $\text{Cpred}_k \neq \infty$ , and  $\text{coins}(hq)$  was moved from  $\text{PenaltyWallet}$ .
- If  $\text{ldc}_k = 1$  and  $k \in C$  and  $\text{Cpred}_k = \infty$  then  $\text{coins}(q)$  was moved from each of  $\text{SimWallet}_j$  for  $j \in C$  and  $\text{coins}(h'q)$  from  $\text{PenaltyWallet}$  and given to  $\mathcal{A}$  (controlling  $P_k$ ).
- If  $\text{ldc}_k = 0$  and  $k \in C$  and  $k+1 \in H$ , then  $\text{coins}(h'q)$  was moved from  $\text{PenaltyWallet}$  and given to  $\mathcal{F}_f^*$ .
- If  $\text{ldc}_k = 0$  and  $k+1 \in C$ , then  $\text{coins}(kq)$  was moved from  $\text{SimWallet}_{k+1}$  and given to  $\mathcal{A}$ .

We now summarize the state of each  $\text{SimWallet}_j$  for  $j \in C$  via the following proposition.

**Proposition 3.** Suppose  $\text{laddep} = 1$  and further assume that  $\mathcal{S}$  does not output  $\text{fail}_{\text{bind}}$  or  $\text{fail}_{\text{hide}}$  during the simulation of the ladder phase. Then for each  $j \in C$  it holds that  $\text{SimWallet}_j$  contains either  $\text{coins}(q)$  or  $\text{coins}(0)$  at the end of the ladder simulation phase. Furthermore,  $\text{PenaltyWallet}$  contains  $\text{coins}(0)$ , and the following hold at the end of the ladder simulation phase:

- if  $n \in H$  and  $\text{ldc}_{j^*} = 1$  for  $j^* \in C$  with  $\text{Cpred}_{j^*} = \infty$ , then for each  $j \in C$  it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ .
- if  $n \in H$  and  $\text{ldc}_{j^*} = 0$  for  $j^* \in C$  with  $\text{Cpred}_{j^*} = \infty$ , then for each  $j \in C$  it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ .
- if  $n \in C$ , then for each  $j \in C \setminus \{n\}$  it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ , and  $\text{SimWallet}_n$  contains  $\text{coins}(0)$ .

*Proof.* We first analyze the case when  $j \in C$  and  $j \neq n$ . If  $\text{ldd}_j = 0$ , then it is easy to see that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$  (since  $\text{laddep} = 1$  and therefore  $\text{rfdep} = 1$ ). Clearly for  $j > 1$ , if  $\text{ldd}_j = 0$ , then  $\text{ldc}_{j-1} = 0$ , i.e., a deposit that was not made cannot be claimed, and so  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ . (Note that  $P_1$  is not required to make a ladder deposit. Note if  $1 \in C$ , then it is easy to see that  $\text{SimWallet}_1$  contains  $\text{coins}(q)$ .) Next consider the case when  $\text{ldd}_j = 1$ . Then, we split the analysis depending on whether

- $\text{ldc}_{j-1} = 1$ : Suppose there exists  $j' \in C$  such that  $\text{Cpred}_{j'} = j$ . (If no such  $j'$  exists, then simply set  $j' = 0$  in the following.) Then we claim that  $\text{ldc}_{j'} = 1$ . First note for all  $i \in H$  such that  $j < i < j'$ , it holds that  $\text{ldc}_i = 1$ . This is because if  $j-1 \in H$ , then  $\text{ldc}_{j-2} = 1$  (otherwise  $\mathcal{S}$  acting as (honest)  $P_{j-1}$  does not make a ladder claim), and so on and so forth. In this case for all  $i \in H$  such that  $j < i < j'$ ,  $\mathcal{S}$  takes  $\text{coins}(q)$  from  $\text{SimWallet}_j$  (and transfers these to  $\text{PenaltyWallet}$ ), and for  $j'$  as above,  $\mathcal{S}$  takes  $\text{coins}(j'q)$  from  $\text{SimWallet}_{j'}$ , and sends these to  $\mathcal{A}$  when  $\mathcal{A}$  acting as  $P_{j'}$  makes the corresponding claim. Now since  $j'q + (j-1-j')q = (j-1)q$ , it follows that  $\text{SimWallet}_j$  now contains  $\text{coins}(q)$ .
- $\text{ldc}_{j-1} = 0$ : There are two cases to handle. First, suppose  $j-1 \in C$ . Then in this case, it is easy to see that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ . This is because since  $P_j$ 's deposit was not claimed, and further, was not distributed to account for  $\text{PenaltyWallet}$  or for  $P_{j-1}$ 's claim. Therefore,  $\mathcal{S}$  simply refunds  $P_j$ 's deposit (i.e., gives  $\text{coins}((j-1)q)$  back to  $\mathcal{A}$ ), and therefore  $\text{SimWallet}_j$  only contains  $\text{coins}(q)$ . The second case is when  $j-1 \notin H$ . In this case,  $\text{ldc}_{j-1} = 0$  implies that  $\text{ldc}_{j-2} = 0$ , and so on until we have  $\text{ldc}_{j'} = 0$  for some  $j' \in C$ . Once again, we have that  $P_j$ 's deposit was not claimed, and further, was not distributed to account for  $\text{PenaltyWallet}$  or for  $P_{j'}$ 's claim. Therefore,  $\mathcal{S}$  simply refunds  $P_j$ 's deposit (i.e., gives  $\text{coins}((j-1)q)$  back to  $\mathcal{A}$ ), and therefore  $\text{SimWallet}_j$  only contains  $\text{coins}(q)$ .

Note that an extra step in the analysis is required for  $j^* \in C \setminus \{n\}$  such that  $\text{Cpred}_{j^*} = \infty$ , as we need to account for  $\text{ldc}_{j^*}$ .

- If  $\text{ldc}_{j^*} = 0$ : In this case, the contents of  $\text{SimWallet}_{j^*}$  remain unchanged in round  $\tau_{j^*}$  (i.e., it contains  $\text{coins}(q)$ ). It remains to be shown that  $\text{PenaltyWallet}$  contains  $\text{coins}(0)$ . Suppose  $\text{contact} = 1$  already before the beginning of round  $\tau_{j^*}$ , then since  $j^* \in C \setminus \{n\}$ , this can happen only if there exists some  $j' \in C$  such that  $j' + 1 \in H$  and  $\text{ldc}_{j'} = 0$ . In this case, it is easy to verify that  $\text{PenaltyWallet}$  contains  $\text{coins}(0)$  as all the coins in  $\text{PenaltyWallet}$  were sent to  $\mathcal{F}_f^*$  by  $\mathcal{S}$ . On the other hand, if  $\text{contact} = 0$  before round  $\tau_{j^*}$ , then once again since  $j^* \in C \setminus \{n\}$ , it must be the case that  $j^* + 1 \in H$ , and as described in the simulation,  $\mathcal{S}$  sends  $\text{coins}(h'q)$  to  $\mathcal{F}_f^*$  where  $\text{coins}(h'q)$  are taken from  $\text{PenaltyWallet}$  and therefore,  $\text{PenaltyWallet}$  contains  $\text{coins}(0)$ .
- Else if  $\text{ldc}_{j^*} = 1$ : In this case,  $\mathcal{S}$  takes  $\text{coins}(q)$  from each of  $\text{SimWallet}_j$  for  $j \in C$  (i.e., including  $j^*$ ) and  $\text{coins}(h'q)$  from  $\text{PenaltyWallet}$  where  $h' = |H'|$  for  $H'$  as in the simulation. As discussed in the description of the ladder phase simulation, the total number of coins from above equals  $\text{coins}(tq) + \text{coins}(h'q) = \text{coins}(j^*q)$  since  $t + h' = j^*$ . Thus at the end of this step, each  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ , and further  $\text{PenaltyWallet}$  contains  $\text{coins}(0)$  as well.

Next we consider the case when  $j = n \in C$ . If  $\text{ldd}_n = 0$ , then the proposition follows. If  $\text{ldd}_n = 1$ , then

again we consider two cases depending on  $\text{ldd}_{n-1}$ . If  $\text{ldd}_{n-1} = 0$ , then by an argument similar to the one made for the case  $\text{ldc}_{j-1} = 0$ , one can see that  $P_n$ 's deposit was neither claimed nor distributed to PenaltyWallet or a corrupt party's claim. Therefore,  $\mathcal{S}$  refunds  $P_n$ 's deposit back to  $\mathcal{A}$ , and therefore,  $\text{SimWallet}_n$  contains  $\text{coins}(0)$ . On the other hand if  $\text{ldc}_{n-1} = 1$ , then it can be verified (following an analysis similar to the case for  $j \in C \setminus \{n\}$  as above, i.e., when  $\text{ldc}_{j-1} = 0$ ) that  $\text{SimWallet}_n$  contains  $\text{coins}(0)$ . (Note that we don't need the extra step in the analysis since  $\text{ldc}_n = 0$  throughout the ladder claim phase.) Thus, we conclude that the proposition holds.  $\square$

This completes the description of the simulation of the ladder claims phase in the case when  $\text{laddep} = 1$ . (Recall that the case  $\text{laddep} = 0$  was handled already.)

Simulating roof claims. By Proposition 3, at this point the following holds:

- If  $n \in C$ , then for each  $j \in C \setminus \{n\}$ , it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ , and  $\text{SimWallet}_n$  contains  $\text{coins}(0)$ .
- If  $n \in H$  and  $\text{ldc}_{n-1} = 1$  (which is equivalent to  $\text{ldc}_{j^*} = 1$  for  $j^*$  as in Proposition 3), then for each  $j \in C$ , it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ . Else if  $n \in H$  but  $\text{ldc}_{n-1} = 0$  (which is equivalent to  $\text{ldc}_{j^*} = 0$  for  $j^*$  as in Proposition 3), then for each  $j \in C$ , it holds that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$ .

Now the simulator acting as  $\mathcal{F}_{\text{CR}}^*$  performs the following: The analysis splits depending on whether  $n \in H$  or not:

- If  $n \in H$ , then
    - if  $\text{ldc}_{n-1} = 1$  then choose  $\text{sh}'_n$  such that  $z \oplus \bigoplus_{j \neq n} \text{sh}'_j = \text{sh}'_n$ , and run the equivocal simulator  $\tilde{S}_2(\text{st}_n, \text{sh}'_n)$  to obtain  $\omega'_n$ . Then set  $\text{Token}'_n = (\text{sh}'_n, \omega'_n)$  and for all  $k \in [n-1]$ , send  $(\text{claim}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, q, W_n)$  to  $\mathcal{A}$  where  $W_n = W_{n-1} \cup \{\text{Token}'_n\}$ .
    - else if  $\text{ldc}_{n-1} = 0$  then  $\mathcal{S}$  sends  $(\text{refund}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, q)$  to  $\mathcal{A}$  for all  $k \in [n-1]$ . Further,  $\mathcal{S}$  sends for all  $k \in C$ , the message  $(\text{refund}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  to  $\mathcal{A}$  controlling  $P_k$  where  $\text{coins}(q)$  is taken from  $\text{SimWallet}_k$ . (Note that after this step,  $\text{SimWallet}_k$  contains no coins.)
  - If  $n \in C$ , then  $\mathcal{S}$  initializes  $H'' = H$  and for all  $k \in [n-1]$  initializes  $\text{rfc}_k = 0$ , and does the following:
    - For each  $k \in [n-1]$ : the simulator waits to receive  $(\text{claim}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, W''_n)$  where  $W''_n = \{\text{Token}''_1, \dots, \text{Token}''_n\}$  from  $\mathcal{A}$  in round  $\tau_n$ .
      - If  $\text{rfdep} = 0$  or  $\text{laddep} = 0$ : If for all  $j \in [n]$ , it holds that  $\text{Verify}(\text{Tag}'_j, \text{Token}''_j) = 1$ , then output  $\text{fail}_{\text{hide}}$  and terminate.
      - Else If  $\text{rfdep} = 1$  and  $\text{laddep} = 1$ : If for all  $j \in [n]$ , it holds that  $\text{Verify}(\text{Tag}'_j, \text{Token}''_j) = 1$ , and yet  $\{\text{Token}'_1, \dots, \text{Token}'_n\} \neq W''_n$ , then output  $\text{fail}_{\text{bind}}$  and terminate.
      - If  $W''_n = \{\text{Token}'_1, \dots, \text{Token}'_n\}$ , then  $\mathcal{S}$  updates  $H'' = H'' \setminus \{k\}$ , and sets  $\text{rfc}_k = 1$ . (Loosely speaking,  $H''$  represents the set of honest parties whose roof deposit was not claimed by malicious  $P_n$ .)
    - If there exists  $k \in [n-1]$  such that  $\text{rfc}_k = 1$ , then  $\mathcal{S}$  sends  $(\text{continue}, \text{sid}, \text{ssid}, H'')$  to  $\mathcal{F}_f^*$  and receives back  $(\text{payback}, \text{sid}, \text{ssid}, \text{coins}((h - h'')q))$  from  $\mathcal{F}_f^*$  where  $h'' = |H''|$ , and adds these  $\text{coins}((h - h'')q)$  to  $\text{SimWallet}_n$ . Then for each  $k$  for which  $\text{rfc}_k = 1$  holds,  $\mathcal{S}$  sends the following:
      - Message  $(\text{claim}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  to  $\mathcal{A}$  controlling  $P_n$  where  $\text{coins}(q)$  are taken from
        - $\text{SimWallet}_n$  if  $k \in H$ , and
        - $\text{SimWallet}_k$  if  $k \in C$ .
      - Message  $(\text{claim}, \text{sid}, \text{ssid}, k, n, \phi_{\text{rf}}, \tau_n, q, W''_n)$  to  $\mathcal{A}$ .
- At the end of this step it holds that for all  $j \in C \setminus \{n\}$  with  $\text{rfc}_j = 1$ ,  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ . Further, for  $n \in C$ , observe that  $\text{SimWallet}_n$  contains  $\text{coins}(0)$ . This is because  $|\{k : k \in H \wedge \text{rfc}_k = 1\}| = (h - h'')$ . Summarizing, we have that  $\text{SimWallet}_j$  contains  $\text{coins}(q)$  for each  $j \in C \setminus \{n\}$  with  $\text{rfc}_j = 0$ , and for all other  $j \in C$ ,  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ .
- Else (i.e., for every  $k \in [n-1]$  it holds that  $\text{rfc}_k = 0$ ),  $\mathcal{S}$  sends  $(\text{abort}, \text{sid}, \text{ssid})$  to  $\mathcal{F}_f^*$ .

- In round  $\tau_n + 1$ ,  $\mathcal{S}$  does the following for each  $k \in [n - 1]$  such that  $\text{rfc}_k = 0$ :
  - If  $k \in H$ , then  $\mathcal{S}$  sends  $(\text{refund}, sid, ssid, k, n, \phi_{\text{rf}}, \tau_n, q)$  to  $\mathcal{A}$ .
  - Else if  $k \in C$ , then  $(\text{refund}, sid, ssid, k, n, \phi_{\text{rf}}, \tau_n, \text{coins}(q))$  to  $\mathcal{A}$  controlling  $P_k$ , where  $\text{coins}(q)$  are taken from  $\text{SimWallet}_k$ .

Note that at the end of this step it holds that for all  $j \in C$ ,  $\text{SimWallet}_j$  contains  $\text{coins}(0)$ .

Finally  $\mathcal{S}$  terminates the simulation, and outputs whatever  $\mathcal{A}$  outputs. This completes the description of the simulator.

**Analysis sketch.** It can be verified from the description of the simulation above that the distribution of coins in the  $\mathcal{F}_{\text{CR}}^*$ -hybrid execution is identical to the distribution of coins in the ideal execution. This allows us to perform the security reductions in the standard way. We construct a sequence of experiments starting with the  $\mathcal{F}_{\text{CR}}^*$ -hybrid execution and ending with the simulated execution and prove that each experiment is indistinguishable from the next.

Experiment 1. This is identical to the real execution of the protocol. We can restate the above experiment with the simulator as follows. We replace the real world adversary  $\mathcal{A}$  with the ideal world adversary  $\mathcal{S}$ . The ideal adversary  $\mathcal{S}$  starts by invoking a copy of  $\mathcal{A}$  and running a simulated interaction of  $\mathcal{A}$  and the honest parties. In this experiment the simulator  $\mathcal{S}$  holds the private inputs and coins of the honest parties and generates messages on their behalf using the honest party strategies as specified by the protocol.

Experiment 2. In this experiment we change how the simulator generates output of the honest parties. In particular, we let  $\mathcal{S}$  extract the input by acting as  $\mathcal{F}_f$ . If the output of  $\mathcal{S}$  is  $\text{fail}_{\text{bind}}$ , then we terminate the simulation. Else,  $\mathcal{S}$  outputs whatever the adversary outputs and terminates the simulation.

First, we claim that the probability that  $\mathcal{S}$  outputs  $\text{fail}_{\text{bind}}$  is negligible in  $\lambda$ . Indeed, this is the case, since an adversary that makes  $\mathcal{S}$  output  $\text{fail}_{\text{bind}}$  can be easily used to break the binding for *honest sender* property of the underlying equivocal commitment scheme. Since we use a secure tag-token scheme, it follows that the probability that  $\mathcal{S}$  outputs  $\text{fail}_{\text{bind}}$  is negligible in  $\lambda$ .

Indistinguishability between experiments 1 and 2 directly follows from (a straightforward hybrid argument involving) the binding property of the commitment scheme.

Experiment 3. In this experiment we change how the simulator  $\mathcal{S}$  generates the first round messages on behalf of the honest parties. In particular instead of committing to the inputs of honest parties  $\mathcal{S}$  just uses the equivocal simulator to generate commitments. If the output of  $\mathcal{S}$  is  $\text{fail}_{\text{hide}}$ , then we terminate the simulation. Else,  $\mathcal{S}$  outputs whatever the adversary outputs and terminates the simulation.

First, we claim that the probability that  $\mathcal{S}$  outputs  $\text{fail}_{\text{hide}}$  is negligible in  $\lambda$ . Indeed, this is the case, since an adversary that makes  $\mathcal{S}$  output  $\text{fail}_{\text{hide}}$  can be easily used to break the hiding property of the underlying equivocal commitment scheme. Since we use a secure tag-token scheme, it follows that the probability that  $\mathcal{S}$  outputs  $\text{fail}_{\text{hide}}$  is negligible in  $\lambda$ .

Indistinguishability between experiments 2 and 3 directly follows from (a straightforward hybrid argument involving) the hiding property of the commitment scheme. In particular there is no need to rewind in the reduction or in the simulation.

Experiment 4. Observe that in experiment 3,  $\mathcal{S}$  uses inputs of honest parties only to obtain the output of the computation. Instead,  $\mathcal{S}$  can obtain the same value by sending extracted input of the adversary to the trusted party.

Note that experiments 3 and 4 are identical. Also observe that experiment 4 is identical to the simulation strategy. This concludes the proof. □

## E Secure Lottery with Penalties

Recall that our notion of fair lottery guarantees the following:

- An honest party never has to pay any penalty.
- The lottery winner has to be chosen uniformly at random.
- If a party aborts after learning whether or not it won the lottery without disclosing this information to honest parties, then every honest party is compensated.

For a formal specification of the ideal functionality see Figure 3. Our protocol proceeds in a similar way to our protocol for secure computation with penalties. Specifically, the parties first engage in a standard secure computation protocol that computes the identity of the lottery winner (i.e., by uniformly selecting an integer from  $[n]$ ), and secret shares this result using pubNMSS (scheme described in Section 3). Now parties need to reconstruct this secret in a fair manner. Note that a malicious party may abort upon learning the outcome of the lottery (say, on learning that it did not win). This is where the fair reconstruction helps, in the sense that parties that did not learn the outcome of the protocol (i.e., the identity of the lottery winner) now receive a penalty payment equal to the lottery prize. However, this alone is not sufficient. One needs to ensure that the lottery winner actually receives the lottery prize too.

Fortunately, by making a minor modification to the “ladder” protocol, we are able to ensure that the lottery winner receives its lottery prize when the reconstruction is completed. Specifically, our modified ladder protocol now has 3 phases: ridge, roof, and ladder phases. The ladder phase is identical to the ladder phase in the fair reconstruction protocol. We now describe at a high level why and how this modification works.

First recall that if parties follow the protocol, then at the end of the ladder claims,  $P_n$  has lost  $(n - 1)q$  coins and every other party has gained  $q$  coins (assuming it can get its roof deposits refunded). That is, effectively party  $P_n$  has “paid”  $(n - 1)q$  coins to learn the outcome of the lottery. Now suppose our roof deposit phase was made w.r.t relations  $\phi_{rf}^j$  by party  $P_j$  such that it pays  $q$  coins to  $P_n$  only if  $P_j$  did not win the lottery.<sup>14</sup> Then, at the end of this phase, it is guaranteed that the lottery winner  $P_j$ , if  $j \neq n$ , has won  $q$  coins, and (only)  $P_n$  has completely paid for the lottery prize. Further even when  $j = n$  (i.e.,  $P_n$  won the lottery) then at the end of the roof deposit phase, party  $P_n$  has only “evened out” and in particular has not won the lottery prize. Effectively,  $P_n$  has paid the lottery prize to the lottery winner.

Of course, such a situation is highly unsatisfactory. We remedy the situation by introducing “ridge” deposits made by each party  $P_j$  except  $P_n$  where  $P_j$  promises to pay its lottery share  $q/n$  to  $P_n$  as long as  $P_n$  reveals all the tokens. This simple fix allows us to prove the following theorem:

**Theorem 4.** *Assuming the existence of one-way functions, there exists a  $n$ -party protocol for secure lottery with penalties in the  $(\mathcal{F}_{OT}, \mathcal{F}_{CR}^*)$ -hybrid model. Further, the protocol requires  $O(n)$  rounds, a total of  $O(n)$  calls to  $\mathcal{F}_{CR}^*$ , and each party is required to deposit  $n$  times the penalty amount.*

A pictorial representation of the phases is shown in Figure 8. (As the protocol is readily obtained by modifying the protocol for secure computation with penalties, we defer a full description of the protocol for lottery with penalties.) The simulation of the lottery protocol closely follows the simulation described in Section D.2. The main difference is that now ridge deposits, claims, and refunds need to be addressed. Fortunately, the ridge deposits, claims, and refunds are handled almost exactly as the roof deposits, claims, and refunds as in Section D.2. Also, in the simulation of the lottery protocol,  $\mathcal{S}$  needs to specify additional sets  $\tilde{H}'$  and  $H''$  to  $\mathcal{F}_f^*$ . These are defined analogously to the definition of  $H''$  in the simulation of Section D.2. Specifically, these sets are required to capture the event when the last party is corrupt, and may choose to claim only some but not all of the roof/ridge deposits.

---

<sup>14</sup>Formally, for  $s \in [n]$ , define  $\phi_{lad}^s(T_1, \dots, T_s) = \phi(\text{Tag}_1, T_1) \wedge \dots \wedge \phi(\text{Tag}_s, T_s)$ . For all  $s \in [n - 1]$ , define  $\phi_{rf}^s(T_1, \dots, T_n) = \phi_{lad}^n(T_1, \dots, T_n) \wedge (\text{Ext}(\text{Tag}_1, T_1) + \dots + \text{Ext}(\text{Tag}_n, T_n) \neq s \bmod n)$ , where  $\text{Ext}$  extracts the exact share (i.e., the input for the commitment) from token  $T$ .

Ridge DEPOSITS.

$$\begin{array}{ccc}
 P_1 & \xrightarrow[T_1 \wedge \dots \wedge T_n]{q/n, \tau_n} & P_n \\
 P_2 & \xrightarrow[T_1 \wedge \dots \wedge T_n]{q/n, \tau_n} & P_n \\
 & \vdots & \\
 P_{n-2} & \xrightarrow[T_1 \wedge \dots \wedge T_n]{q/n, \tau_n} & P_n \\
 P_{n-1} & \xrightarrow[T_1 \wedge \dots \wedge T_n]{q/n, \tau_n} & P_n
 \end{array}$$

Roof DEPOSITS.

$$\begin{array}{ccc}
 P_1 & \xrightarrow[\phi_{\text{rf}}^1]{q, \tau_n} & P_n \\
 P_2 & \xrightarrow[\phi_{\text{rf}}^2]{q, \tau_n} & P_n \\
 & \vdots & \\
 P_{n-2} & \xrightarrow[\phi_{\text{rf}}^{n-2}]{q, \tau_n} & P_n \\
 P_{n-1} & \xrightarrow[\phi_{\text{rf}}^{n-1}]{q, \tau_n} & P_n
 \end{array}$$

Ladder DEPOSITS.

$$\begin{array}{ccc}
 P_n & \xrightarrow[T_1 \wedge \dots \wedge T_{n-1}]{(n-1)q, \tau_{n-1}} & P_{n-1} \\
 P_{n-1} & \xrightarrow[T_1 \wedge \dots \wedge T_{n-2}]{(n-2)q, \tau_{n-2}} & P_{n-2} \\
 & \vdots & \\
 P_3 & \xrightarrow[T_1 \wedge T_2]{2q, \tau_2} & P_2 \\
 P_2 & \xrightarrow[T_1]{q, \tau_1} & P_1
 \end{array}$$

Figure 8: Ridge, Roof, and Ladder deposit phases for fair lottery.

## F Realization of $\mathcal{F}_{\text{CR}}^*$ via Bitcoin

Following [78, 11], an implementation of the ideal functionality  $\mathcal{F}_{\text{CR}}^*$  via Bitcoin is shown in Figure 9.

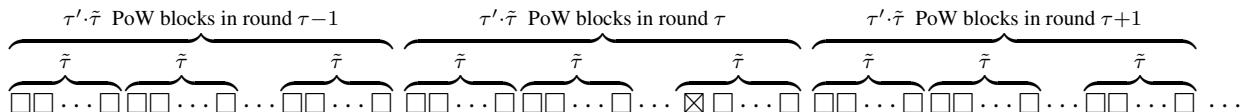
The intended purpose of  $\mathcal{F}_{\text{CR}}^*$  is to let party  $P_s$  deposit an  $x$  amount of its coins while specifying a circuit  $\phi_r$ , so that if party  $P_r$  reveals a witness  $w_r$  that satisfies  $\phi_r(w_r) = 1$  in round  $\tau$ , then  $P_r$  obtains the  $x$  coins that  $P_s$  deposited. Otherwise,  $P_s$  recovers its  $x$  coins in round  $\tau + 1$ .

Hence,  $\mathcal{F}_{\text{CR}}^*$  is a powerful tool for the purpose of a “fair” exchange of values, where the fairness is in terms of penalties. That is, if  $P_1$  seeks to obtain  $w_2$  and  $P_2$  seeks to obtain  $w_1$ , they can engage in two concurrent  $\mathcal{F}_{\text{CR}}^*$  invocations in which each of them deposits  $x$  coins, with the purpose of exchanging  $w_1$  for  $w_2$ . An honest party  $P_i$  who reveals its  $w_i$  will collect the  $x$  coins of the other party; if the other party aborts after learning  $w_i$  then it will be penalized, because the honest party will also collect its own  $x$  coins at the next round. The usefulness of  $\mathcal{F}_{\text{CR}}^*$  is better understood when compared to a Bitcoin-based commitment functionality, i.e. to a primitive with which the party  $P_s$  who deposits the  $x$  coins is required to reveal its own witness  $w_s$  at round  $\tau$ , or else it will forfeit the coins to  $P_r$ . The weakness of such a commitment functionality is that in case an honest  $P_1$  commits to  $w_1$  first (expecting  $P_2$  to commit to  $w_2$ ),  $P_1$  will be forced to reveal  $w_1$  if  $P_2$  aborts. This implies that a commitment functionality is useful for exchanging fresh random values (as in the lottery protocol of [4, Section IV]), while  $\mathcal{F}_{\text{CR}}^*$  can be used to exchange any values, and in particular values that depend on an earlier round of the protocol.

The Bitcoin mechanisms that we take advantage of in order to realize  $\mathcal{F}_{\text{CR}}^*$  are *timelock*, scripts, and the manner in which transactions are chained (familiarity with Appendix G is hereby assumed). To elaborate, the Bitcoin realization of  $\mathcal{F}_{\text{CR}}^*$  in Figure 9 is done according to the following outline. First,  $P_s$  creates a transaction  $txn_{\text{CR}}$  that takes an input of  $x$  coins that it controls, and can be redeemed according to “( $P_s$ ’s signature AND  $P_r$ ’s signature) OR ( $w_r$  for which  $\phi_s(w_r) = 1$  AND  $P_r$ ’s signature)”.  $P_s$  keeps  $txn_{\text{CR}}$  private, and creates another transaction  $txn_{\text{refund}}$  that spends  $txn_{\text{CR}}$  to an output it controls, but has locktime set in the future. Then,  $P_s$  sends  $txn_{\text{refund}}$  to  $P_r$ , asking  $P_r$  to sign it. Notice that since  $P_r$  sees only the hash of  $txn_{\text{CR}}$ , it cannot broadcast it to the Bitcoin network and thus extort  $P_s$  or cause  $P_s$  to lose its  $x$  coins. Now,  $P_r$  signs the timelocked transaction  $txn_{\text{refund}}$ , and sends the signature to  $P_s$ . At this point,  $P_s$  can be sure that it will be able to recover its  $x$  coins after the locktime expires, by attaching  $P_r$ ’s signature and its own signature to  $txn_{\text{refund}}$ , thereby meeting the condition “( $P_s$ ’s signature AND  $P_r$ ’s signature)” of  $txn_{\text{CR}}$ .

Let us note that this realization of  $\mathcal{F}_{\text{CR}}^*$  indeed captures the automatic refunds in the last phase of the ideal functionality. In particular, if both parties are corrupt and abort after the *Deposit* phase, then the  $x$  coins could now be redeemed by either party (so the coins are now under the control of the adversary).

The security of this  $\mathcal{F}_{\text{CR}}^*$  realization and other Bitcoin-based realizations relies on resistance to double-spending attacks. This implies that it can be helpful to differentiate between a normal round in which parties send messages to other parties, and a “Bitcoin round” in which parties may also broadcast transaction messages to the Bitcoin network. The rationale for this is that a “Bitcoin round” can be granular and consist of multiple normal rounds, because of the waiting period that is required in order to be protected (w.h.p.) from double-spending attacks (see Appendix G for concrete details). In this work, we do not make this distinction in the ideal functionalities, but we will put this kind of granularity to use in the Bitcoin-based realizations. Hence, we parameterize the  $\mathcal{F}_{\text{CR}}^*$  realization by two parameters. The parameter  $\tilde{\tau}$  represents the double-spending safety interval, i.e. the number of PoW blocks that need to be solved until an honest party becomes confident enough that a transaction will not be reversed. The parameter  $\tau'$  represents the number of such intervals in a single round. Let us illustrate it as follows:



In the  $\mathcal{F}_{\text{CR}}^*$  realization, an honest  $P_s$  will attempt to redeem  $txn_{\text{refund}}$  at the marked block  $\boxtimes$ , i.e.  $\tilde{\tau}$  blocks prior to

$\mathcal{F}_{\text{CR}}^*$  realized by Bitcoin

- *Deposit phase.*

- $P_s$  requests a fresh public key by sending  $(\text{deposit\_init}, sid, ssid, s, r, \tau)$  to  $P_r$ .
- $P_r$  generates a fresh  $(sk_r, pk_r)$  pair and sends  $(\text{deposit\_ack}, sid, ssid, s, r, \tau, pk_r)$  to  $P_s$ .
- $P_s$  takes any public key  $pk_s$  for which only it knows the corresponding  $sk_s$ , as well as an arbitrary circuit i.e. any Bitcoin script  $\phi(\cdot)$ , and creates a Bitcoin transaction  $txn_{\text{CR}}$  that redeems  $x$  coins that it controls to the following  $\pi$  output script:  $\pi(\cdot) \triangleq \text{OP\_CHECKSIG}(pk_r, \cdot) \text{ AND } (\text{OP\_CHECKSIG}(pk_s, \cdot) \text{ OR } \phi_r(\cdot))$ .
- $P_s$  computes  $id_{\text{CR}} = \text{SHA256d}(txn_{\text{CR}})$  and prepares a transaction  $txn_{\text{refund}}$  that takes  $id_{\text{CR}}$  as its input script, has locktime of  $\tau \cdot \tau' \cdot \tilde{\tau}$  blocks, and spends the output of  $txn_{\text{CR}}$  to some output script  $\pi'(\cdot)$  that it controls, i.e. the simplified form of  $txn_{\text{refund}}$  is  $txn_{\text{refund}}^{\text{simp}} \triangleq ((id_{\text{CR}}, 1), [(x, \pi')], b_{\text{curr}} + \tau \cdot \tau' \cdot \tilde{\tau})$  where  $b_{\text{curr}}$  is the current height of the longest extension of the blockchain that  $P_s$  is aware of.
- $P_s$  sends  $(\text{deposit\_sign}, sid, ssid, s, r, txn_{\text{refund}}^{\text{simp}})$  to  $P_r$ .
- $P_r$  sends  $(\text{deposit\_sign\_ack}, sid, ssid, s, r, sig_r = \text{Sign}_{sk_r}(txn_{\text{refund}}^{\text{simp}}))$  to  $P_s$ .
- $P_s$  ensures that  $\text{Vrfy}_{pk_r}(txn_{\text{refund}}^{\text{simp}}, sig_r) = 1$ , and then broadcasts  $txn_{\text{CR}}$  to the Bitcoin network.

- *Claim phase.* After  $\tau \cdot \tau' \cdot \tilde{\tau}$  blocks have been solved by the Bitcoin network:

- $P_r$  broadcasts to the Bitcoin network a transaction that redeems  $txn_{\text{CR}}$  to another address (output script) that it controls, by providing  $\text{Sign}_{sk_r}(txn_{\text{CR}}^{\text{simp}})$  and revealing a witness  $w_r$  that satisfies  $\phi_r(w_r) = 1$ .

- *Refund phase.* After  $(\tau + 1) \cdot \tau' \cdot \tilde{\tau} - \tilde{\tau}$  blocks have been solved by the Bitcoin network:

- $P_s$  computes  $\text{Sign}_{sk_s}(txn_{\text{refund}}^{\text{simp}})$  and combines it with  $sig_r$  into an input script  $w_s$  that redeems  $txn_{\text{CR}}$ , then inserts  $w_s$  into  $txn_{\text{refund}}^{\text{simp}}$  and broadcasts the now complete  $txn_{\text{refund}}$  to the Bitcoin network.

*Remarks:*

- Since  $P_r$  only sees the  $txid$  hash of the  $txn_{\text{CR}}$  transaction, it generates a fresh  $(sk_r, pk_r)$  pair to protect itself from being tricked into signing a malevolent transaction that steals its other coins [11].
- The parameters  $\tau', \tilde{\tau}$  correspond to the double-spending safety distance (see further details in Appendix F).

Figure 9: Implementation of the ideal functionality  $\mathcal{F}_{\text{CR}}^*$  via Bitcoin.

round  $\tau + 1$ . This guarantees that either  $P_s$  will succeed in recovering its coins( $x$ ) deposit, or else  $P_r$  revealed  $w_r$  until round  $\tau + 1$ . In other words, in order to realize the ideal functionality  $\mathcal{F}_{\text{CR}}^*$  while avoiding race conditions,  $P_s$  broadcasts the refund message in the last interval of round  $\tau$ , rather than in the beginning of round  $\tau + 1$ .

Let us point out that the coins that a party  $P_s$  uses in one invocation of our realization of  $\mathcal{F}_{\text{CR}}^*$  are wholly separate from the coins that  $P_s$  can use in another concurrent invocation. This is true since otherwise one of the  $txn_{\text{CL}}$  transactions that  $P_s$  broadcasts to the Bitcoin network will be rejected as invalid - prior to the round in which  $P_r$  redeems the  $txn_{\text{CL}}$  transaction - as the coins will have already been spent.

Finally, we note that in case a corrupt  $P_r$  reveals  $w_r$  by broadcasting  $txn_{\text{CR}}$  before round  $\tau$ , the honest  $P_s$  can simply wait until round  $\tau$  and then continue to execute the protocol.

## G Outline of the Bitcoin protocol

Bitcoin is a decentralized network in which *miner* nodes engage in *Proof of Work* (PoW) computations to create a chain of blocks (a.k.a. “blockchain”), whose purpose is to synchronize the money transactions of that users in this network. The blocks keep being generated at predictable time intervals, as the honest (either altruistic or

self-interested) nodes follow this protocol outline:

- Miners collect transactions that are broadcasted over the network, and try to generate a block via repeated invocations of a hash function on data that consists of the transactions that they see fit to include, the hash of the previous block, a public key, and a nonce.
- When a miner succeeds in generating a block, meaning that the hash of its block data is smaller than the current difficulty target, it broadcasts its block to the network.
- When other miners see that this block is valid, i.e. it references the hash of the previous block and meets the current difficulty target, and see that it is the longest<sup>15</sup> extension of the blockchain that they are aware of, they move on to continue to extend the blockchain from this block.
- The block reward (newly minted coins) and the fees from the transactions that the miner collected go to the public key that the miner provided. This means that only this miner can spend the coins that it earned, by signing with its corresponding secret key.
- The difficulty level readjusts according to the mining power that participates, by updating the hash target value every 2016 blocks (approximately 2 weeks) so that blocks get generated once every 10 minutes on average.

In fact, the transactions that the Bitcoin protocol allows are not limited to the secret key / public key option that we specified above. Instead, Bitcoin supports a scripting language that is quite comprehensive, though not Turing-complete (mainly to avoid denial of service attacks). Thus, an arbitrary circuit  $\pi(\cdot)$  can replace the role of the public key, so that a witness  $w$  which satisfies  $\pi(w) = 1$  replaces the role of the secret key.

The usefulness of the Bitcoin network arises from ability of its users to form transactions as they please, i.e. to transfer (fractional amounts of) coins that originated as a reward that the miners earned.

In its general form, each Bitcoin transaction consists of  $n \geq 1$  inputs,  $k \geq 1$  outputs, and a *locktime* value  $\tau$ :

$$\left( [(t_1, i_1, w_1), \dots, (t_n, i_n, w_n)], [(y_1, \pi_1), \dots, (y_k, \pi_k)], \tau \right),$$

where  $t_j$  is an *id* (SHA256d<sup>16</sup> hash) of a previous transaction  $T_j$ ,  $i_j$  is the  $j^{\text{th}}$  output  $(x_j, \pi'_j)$  of  $T_j$ , and  $w_j$  is a witness (input script) that should satisfy the output script  $\pi'_j$  of the  $j^{\text{th}}$  output of  $T_j$ . The transaction is valid if:

1. It is added to the blockchain after time  $\tau$  (the value  $\tau$  can be specified either in blocks or in seconds).
2. For all  $j = 1, \dots, n$ , it holds that  $\pi'_j(w_j) = 1$ .
3. For all  $j = 1, \dots, n$ , the output  $(x_j, \pi'_j)$  has not already been spent by another transactions.
4.  $\sum_{j=1}^n y_j \leq \sum_{j=1}^n x_j$ , and the difference  $\sum_{j=1}^n x_j - \sum_{j=1}^n y_j$  is a fee that goes to the miner who creates a block that includes this transaction.

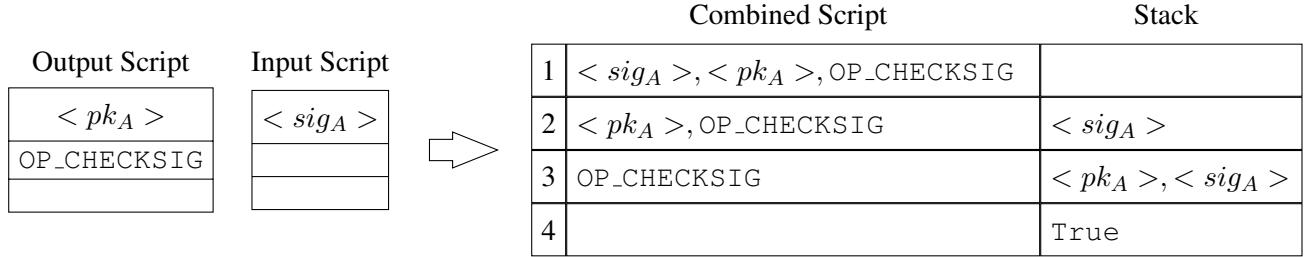
Bitcoin script opcodes are allowed to take a simplified form of the transaction itself as an implicit parameter. The simplified form of a transaction is  $([(t_1, i_1), \dots, (t_n, i_n)], [(y_1, \pi_1), \dots, (y_k, \pi_k)], \tau)$ , i.e. it excludes all its input scripts  $w_j$ . In particular, `OP_CHECKSIG(pk, sig)` in an opcode that invokes ECDSA signature verification where the plaintext message  $m$  is the simplified transaction itself as an implicit argument, and the two explicit arguments  $pk, sig$  are used to verify whether  $sig$  is valid signature of  $m$  under the public key  $pk$ . The simplified form is needed because an input script  $w_j$  may contain a signature for the simplified transaction (and therefore the signature cannot be a part of it), and it is justified because the operation that nobody besides the holder of the secret key should be able to do is specifying (i.e. signing) the new output scripts for the coins that were controlled by that secret key.

---

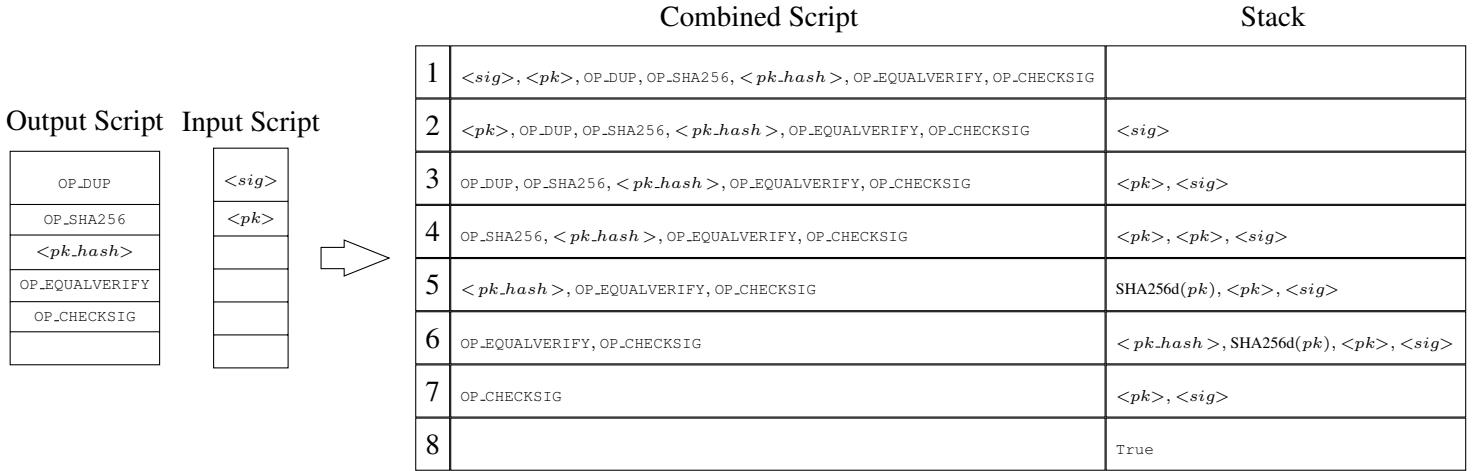
<sup>15</sup>Longest not in total number of blocks, but measured in work difficulty without distinction between blocks that belong to the same retarget window of 2016 blocks.

<sup>16</sup> $\text{SHA256d}(\cdot) \triangleq \text{SHA256}(\text{SHA256}(\cdot))$

To demonstrate this, let us show how the combination of an output script  $\pi'_j$  and an input script  $w_j$  is used to test whether  $\pi'_j(w_j) = 1$ . The contents of the input and output scripts are combined into a single script, which then executes. During the execution of the script, non-executable opcodes are pushed to the top of the data stack, and executable opcodes pop elements from this stack as their arguments. Hence, the  $\text{OP\_CHECKSIG}(pk_A, \cdot)$  operation that we invoke in the Bitcoin realizations of our ideal functionalities works as follows:



Let us also give a variation of the most common kind of a Bitcoin transaction as an example. Let  $\pi'_j$  be an output script that has some  $pk\_hash$  value hardcoded inside of it, so that it can be redeemed via an input script  $w_j = (pk, sig)$  for which  $\text{SHA256d}(pk) = pk\_hash$  holds, and  $sig$  is a valid signature for the simplified transaction under the public key  $pk$ . By using an  $\text{OP\_DUP}$  opcode that duplicates the value at the top of the stack, these scripts can be implemented as follows:



**Note regarding hash-based commitments.** In our practical Bitcoin-based realizations, we rely on the fixed hash function  $\text{SHA256d}$  to implement the  $\text{Hash}(m \parallel \omega)$  commitments heuristically (c.f. Appendix D). This is well motivated, because an attack on  $\text{SHA256d}$  implies that the underlying Bitcoin infrastructure that we utilize becomes insecure.<sup>17</sup>

**Security analysis.** The security of the Bitcoin network is sustained under the assumption that the majority of the PoW mining power follows the protocol. An individual Bitcoin user becomes increasingly confident that a payment transaction will not be reversed (as it could then be double-spent) when additional PoW blocks extend the block in which that transaction resides. The probability of a successful double-spending attack can be analysed by examining a random-walk variant that is known as the “gambler’s ruin” stochastic process [84, 92]. The financial cost of an attack depends on the amount of honest PoW mining power that participates. For

<sup>17</sup>See <https://bitcointalk.org/index.php?topic=120473.msg1301958#msg1301958>.

example, with  $\mathcal{F}_{\text{CR}}^*$  a corrupt party  $P_s$  can try to double-spend the  $txn_{\text{CR}}$  transaction after  $P_r$  redeems  $txn_{\text{CR}}$  by revealing a witness  $w_r$  that satisfies  $\pi(w_r) = 1$ , but this would require  $P_s$  to reverse  $\tau \cdot \tau' \cdot \tilde{\tau}$  PoW blocks. It is highly costly and unlikely that a successful double-spending attack will be carried out, for example 0.1% success probability if the expected time of one round is 60 minutes ( $\tau = 1$  and e.g.  $\tau' = 2, \tilde{\tau} = 3$ ) and  $P_s$  controls 10% of the PoW mining power [84, 92]. Therefore, the parties may regard the decentralized Bitcoin network as a kind of a trusted dealer, i.e. a trusted dealer that supports special coin-aware functionalities of the kind that is presented in this work.

If the financial value of the outputs of the secure computation (or the size of the lottery pot) is relatively low, the parties may even resort to rounds in which they do not wait for the next PoW block to be solved. Such rounds do not use the PoW irreversibility property, and instead the mining race degrades into a network race. This is similar to Point of Sale for low value Bitcoin transactions, as merchants can take a small risk by accepting transactions with no PoW confirmations, while they listen on the network to detect double-spending attempts. However, a double-spending attack that offers a higher fee to honest but self-interested miners becomes much easier in this case.

One way to look at how we achieve fairness in Bitcoin-based protocols is as follows. We take advantage of the irreversibility property that the computational power devoted to *Proof of Work* in the Bitcoin network lends us, thereby allowing our protocols to function in a way that is similar to that of time-based (gradual release) protocols. However, the parties in our protocols only need to have stake in game, by giving security deposits which they may later reclaim (dependant upon their execution of the protocol), rather than carrying out intensive computational tasks (in security parameter number of rounds) themselves.

**Note regarding transaction fees.** Secure computation with penalties requires that honest parties never lose coins. When we realize the ideal functionalities via Bitcoin, it is typically the case that the parties need to send some sort of a security deposit, and honest parties should fully recover their security deposit before the protocol terminates. However, miners may require small fees for the transactions that deal with these security deposits. For example, with  $\mathcal{F}_{\text{CR}}^*$ , an honest party  $P_s$  may ask a corrupt party  $P_r$  to reveal a secret value  $w_r$  until round  $\tau$ , therefore a fee might be required both for the transaction in which  $P_s$  makes the security deposit of  $x$  coins, and for the transaction in which  $P_s$  recovers its  $x$  coins in round  $\tau + 1$ . At the current state of the Bitcoin network, the block reward subsidy of newly minted coins implies that fees are not required, except for differentiating between legitimate transactions and “spam” transactions that may saturate the network and bloat the blockchain. In the future, fees will be required, though one of the main goals of a cryptocurrency is to allow users to transact with low fees. Hence, we cannot claim that a Bitcoin-based system is able to fully realize our ideal functionalities, but rather that the fees that such a system requires may be negligible. Further, it is possible to devise mixed *Proof of Work* and *Proof of Stake* systems in which the users do not pay fees, though coin holders will see their purchasing power erode (or not appreciate as much as it would have otherwise) due to monetary inflation (c.f. [21]).

**Ideal versus actual Bitcoin-based systems.** There are certain differences between the current Bitcoin implementation, and an implementation that would be that most suitable for the ideal functionalities that we present. For one, the ECDSA signatures scheme that is deployed in Bitcoin gives a fixed number of  $\kappa = 128$  bits of security. It could be possible to extend the Bitcoin script language by adding an opcode for a digital signatures algorithm with a variable-length security parameter, though this may open the door to denial of service attacks. It is typically not enough to simply require a higher fee when such a security parameter is greater, because the risk of network DoS attacks implies that the nodes that propagate the transaction (and do not receive the fee) must verify it before re-broadcasting it. Hence it is preferable to have a cap on the transaction size, and in addition bound the verification time with a small polynomial function of the transaction size. In any case, it is indeed simple to add a new opcode for a signatures algorithm with a fixed security parameter  $\kappa' > 128$ , and thereby achieve an adequate level of security in a heuristic sense.

Another issue is that Bitcoin transactions are susceptible to malleability attacks, i.e. the purportedly unique id hash of a transaction could be mutated while the signature for the unmutated version is still valid. As discussed

above, since only the simplified form of the transaction can be signed, any mutation to an input script  $w_j$  that still satisfies  $\pi'(w_j) = 1$  will exhibit this behavior, and unfortunately such mutations exist w.r.t. the Bitcoin opcodes. This problem implies that the locktime mechanism that is used to realize functionalities such as  $\mathcal{F}_{\text{CR}}^*$  is vulnerable to malleability attacks, because the id of the refund transaction is invalidated (c.f. [11, 4, 3]). The malleability issue is basically a bug<sup>18</sup>, which should be resolved by eliminating all the possible sources of mutations.

Hence, when we discuss Bitcoin realizations in this work, we assume that the Bitcoin system is ideal in this regard, since there is nothing inherently restrictive in these kinds of problems.

---

<sup>18</sup>Further information is at [https://en.bitcoin.it/wiki/Transaction\\_Malleability](https://en.bitcoin.it/wiki/Transaction_Malleability). There are practical ways to mitigate the problem while the bug is still in place, e.g. <https://bitcointalk.org/index.php?topic=303088.0>.