How to Use Bitcoin to Design Fair Protocols

Iddo Bentov and Ranjit Kumaresan

Department of Computer Science, Technion, Haifa, Israel
{idddo|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 $\mathcal{F}^{\star}_{\mathrm{CR}}$ (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 $\mathcal{F}^{\star}_{\mathrm{CR}}$ functionality, and is otherwise independent of the Bitcoin ecosystem. We also show an efficient realization of $\mathcal{F}^{\star}_{\mathrm{CR}}$ 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 $\mathcal{F}^{\star}_{\mathrm{CR}}$ 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 [50, 30, 12, 19] of this seemingly impossible-to-achieve notion has been one of the most striking contributions of modern cryptography. However, definitions of secure computation [29] do vary across models, in part owing to general impossibility results for fair coin-tossing [22]. 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.[1] 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.

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 [14, 22]. 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 [15, 40]) 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 [36, 43] 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.[2] 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 adver-

[1] 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.

[2] 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 [44, 35].

sary. 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 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.[3]

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 [42, 41, 10]. 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 [47] is a peer-to-peer network that uses the power of cryptography to emulate (among other things) a trusted bank. 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 [21, 2, 6].

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 [2, 42, 41, 1] 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

[3] This is especially true in today's world where cheap digital pseudonyms [23] are available.

Bitcoin network (e.g., Litecoin, PayPal, or a central trusted bank).[4] Our main contributions include providing formal definitions and efficient realizations for:

- **Claim-or-refund functionality $\mathcal{F}^{\star}_{\mathrm{CR}}$.** 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. In the full version of our paper [13], we describe a Bitcoin protocol for realizing this functionality that requires parties to make only two transactions on the Bitcoin network. We note that variants of $\mathcal{F}^{\star}_{\mathrm{CR}}$ have been constructed and used in [45, 7, 6].
- **Secure computation with penalties $\mathcal{F}^{\star}_{f}$.** 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 $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}^{\star}_{\mathrm{CR}})$-hybrid model that requires only $O(n)$ rounds[5] and $O(n)$ calls to $\mathcal{F}^{\star}_{\mathrm{CR}}$.
- **Secure lottery with penalties $\mathcal{F}^{\star}_{\mathrm{lot}}$.** 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 $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}^{\star}_{\mathrm{CR}})$-hybrid model that requires only $O(n)$ rounds and $O(n)$ calls to $\mathcal{F}^{\star}_{\mathrm{CR}}$.

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 somewhat 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 [6] and Andrychowicz *et al.* [2, 1]. Indeed, our work is heavily inspired by [6, 2] 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 [2] require quadratic number of transactions to be made on the Bitcoin network. In contrast our protocols require only a linear number of Bitcoin transactions. (See full version for a more detailed comparison with [2].) In a followup work [1] that is concurrent to and independent of ours, the authors of [2] 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

[4] Indeed, we can readily adapt our constructions to obtain the first multiparty solutions enjoying "legally enforceable" fairness [42].

[5] Contrast this with the gradual release mechanism which require security parameter number of rounds even when $n = 2$.

the *multiparty* setting. As far as fair two-party secure computation is concerned, although the goal of [1] and ours is the same, the means to achieve the goal are significantly different. Specifically, the protocols of [2, 1] 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.

- *Fairness in standard secure computation.* Fair two party coin tossing was shown to be impossible in [22]. Completely fair secure computation for restricted classes of functions was shown in [32, 3], while partially fair secure computation for all functions were constructed in [34, 9]. Complete primitives for fairness were extensively studied in [33].
- *Gradual release mechanisms.* Starting from early works [8, 31], gradual release mechanism have been employed to solve the problem of fair exchange in several settings [14, 24, 28]. A good survey of this area can be found in [49]. A formal treatment of gradual release mechanisms can also be found in [27].
- *Optimistic model.* There has been a huge body of work starting from [5, 4, 11] that deals with optimistic models for fair exchange (e.g., [41, 46, 25]). Optimistic models for secure computation was considered in [15]. [41] consider a model similar to ours where receiving payment in the event of breach of fairness is also considered fair.
- *Legally enforceable fairness.* Chen, Kudla, and Paterson [20] designed protocols for fair exchange of signatures in a model where signatures are validated only in a court-of-law. Following this, Lindell [42] 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).

2 Models and Definitions

Before we begin, we note that our formalization is heavily inspired by prior formalizations in settings similar to ours [42, 27]. Let n denote the number of parties and t (resp. h) denote the number of corrupted (resp. honest) parties. We consider settings where $t < n$.[6] 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*

[6] 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.

one another. Further we assume that each party has its own *wallet* and *safe*.[7] All its coins are distributed between its wallet and its 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 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 [42]).

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

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 [17] of our ideal functionalities.
  - The first type of ideal functionalities are standard ideal functionalities used in secure computation literature These functionalities only provide security with agreement on abort [29]. 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 $\mathcal{F}^{\star}_{\mathrm{xxx}}$ (i.e., with superscript $\star$) 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 [27, 38] on how to make the relevant modifications about synchrony assumptions in the UC-framework [17].
- Our special ideal functionality $\mathcal{F}^{\star}_{\mathrm{CR}}$ 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 [27].

[7] The distinction between wallet and safe will become clear in the description of the ideal/real processes.

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 [18]. To avoid this, one may design concurrently secure protocols based only on pure complexity-theoretic assumptions. 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).[8] Further, the standalone variant of our protocols require no such setup.

Preliminaries. A function $\mu(\cdot)$ is negligible in λ if for every positive polynomial $p(\cdot)$ and all sufficiently large λ'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 $X \overset{c}{\equiv} Y$ 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 λ. We follow standard definitions of secure computation [29]. 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 "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, 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.

[8] Also note, in practice, one may obtain heuristic UC security in the programmable random oracle model.

Note on the other hand, we give the real/ideal 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)[9] 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 $\textsc{ideal}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)$ 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 λ. 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 π in such a model with $\mathcal{A}$ by $\textsc{hybrid}^{g}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$, where z denotes $\mathcal{Z}$'s input. We are now ready to define what it means for a protocol to $\mathsf{SCC\ realize}$ a functionality.

Definition 1. *Let $n \in \mathbb{N}$. Let π 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 π $\mathsf{SCC\ realizes}$ $\mathcal{G}_f$ 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 π 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 $\mathcal{Z}$,*

$$\{\textsc{ideal}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \overset{c}{\equiv} \{\textsc{hybrid}^{g}_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}.$$

$\diamondsuit$

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 [17]). For the results in this paper, we only need to *assume* that the Bitcoin protocol realizing $\mathcal{F}^{\star}_{\mathrm{CR}}$ is concurrently composable. Other than this, we require only standard sequential composition [16]. 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.

[9] I.e., we implicitly give $\mathcal{Z}$ the power to create new coins.

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

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

2.1 Special ideal functionalities

Ideal functionality $\mathcal{F}^{\star}_{\mathrm{CR}}$. 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, $\mathcal{F}^{\star}_{\mathrm{CR}}$ allows a sender P_s to *conditionally* send $\mathsf{coins}(x)$ 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 τ, 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 $\mathcal{F}^{\star}_{\mathrm{CR}}$.

The importance of the above functionality is a highly efficient realization via *Bitcoin* that requires only two transactions to be made on the network. See full version [13] for more details. In the Bitcoin realizations of the ideal functionalities, sending a message with $\mathsf{coins}(x)$ 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.

$\mathcal{F}^{\star}_{\mathrm{CR}}$ with session identifier sid, running with parties $P_1, \ldots, P_n$, a parameter 1^{λ}, and an ideal adversary $\mathcal{S}$ proceeds as follows:

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

Fig. 1. The special ideal functionality $\mathcal{F}^{\star}_{\mathrm{CR}}$.

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

$\mathcal{F}_f^\star$ with session identifier sid running with parties $P_1, \ldots, P_n$, a parameter 1^λ, 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 $(\mathsf{input}, sid, ssid, r, y_r, \mathsf{coins}(d))$ from P_r for all $r \in H$. Then wait to receive a message $(\mathsf{input}, sid, ssid, \{y_s\}_{s \in C}, H', \mathsf{coins}(h'q))$ from $\mathcal{S}$ where $h' = |H'|$.
- *Output phase:*
  - Send $(\mathsf{return}, sid, ssid, \mathsf{coins}(d))$ to each P_r for $r \in H$.
  - Compute $(z_1, \ldots, z_n) \leftarrow f(y_1, \ldots, y_n)$.
    - If $h' = 0$, then send message $(\mathsf{output}, sid, ssid, z_r)$ to P_r for $r \in [n]$, and terminate.
    - If $0 < h' < h$, then send $(\mathsf{extra}, sid, ssid, \mathsf{coins}(q))$ to P_r for each $r \in H'$, and terminate.
    - If $h' = h$, then send message $(\mathsf{output}, sid, ssid, \{z_s\}_{s \in C})$ to $\mathcal{S}$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid, H'')$, then send $(\mathsf{output}, sid, ssid, z_r)$ to P_r for all $r \in H$, and send $(\mathsf{payback}, sid, ssid, \mathsf{coins}((h - h'')q))$ to $\mathcal{S}$ where $h'' = |H''|$, and send $(\mathsf{extrapay}, sid, ssid, \mathsf{coins}(q))$ to P_r for each $r \in H''$.
  - Else if $\mathcal{S}$ returns $(\mathsf{abort}, sid, ssid)$, send $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q))$ to P_r for all $r \in H$.

Fig. 2. The special ideal functionality $\mathcal{F}_f^\star$ for secure computation with penalties.

- 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 $\mathcal{F}_f^\star$ for secure computation with penalties in Figure 2. We elaborate more on the definition of the ideal functionality $\mathcal{F}_f^\star$ below.

Ideal functionality $\mathcal{F}_f^\star$. In the first phase, the functionality $\mathcal{F}_f^\star$ receives inputs for f from all parties. In addition, $\mathcal{F}_f^\star$ allows the ideal world adverary $\mathcal{S}$ 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 $\mathsf{coins}(d)$ in the input phase.[10,11] Then, in the output phase, $\mathcal{F}_f^\star$ returns the deposit made by honest parties back to them. If insufficient 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

[10] Ideally, we wouldn't want an honest party to deposit any coins, but we impose this requirement for technical reasons.

[11] 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.

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 $\mathcal{S}$.

$\mathcal{F}^{\star}_{\mathrm{lot}}$ with session identifier sid running with parties $P_1, \ldots, P_n$, a parameter 1^λ, 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 $(\mathsf{input}, sid, ssid, r, \mathsf{coins}(d))$ from P_r for all $r \in H$. Then wait to receive a message $(\mathsf{input}, sid, ssid, \{y_s\}_{s \in C}, H', \mathsf{coins}(h'q + (tq/n)))$ from $\mathcal{S}$ where $h' = |H'|$.
- *Output phase:* Choose $r^* \leftarrow_R \{1, \ldots, n\}$.
  - If $h' = 0$, then send message $(\mathsf{output}, sid, ssid, r^*)$ to P_r for $r \in [n]$, and message $(\mathsf{return}, sid, ssid, \mathsf{coins}(d - q/n))$ to each P_r for $r \in H$. and message $(\mathsf{prize}, sid, ssid, \mathsf{coins}(q))$ to P_{r^*}, and terminate.
  - If $0 < h' < h$, then send $(\mathsf{extra}, sid, ssid, \mathsf{coins}(q))$ to P_r for each $r \in H'$, and message $(\mathsf{return}, sid, ssid, \mathsf{coins}(d))$ to each P_r for $r \in H$, and send $(\mathsf{sendback}, sid, ssid, \mathsf{coins}(tq/n))$ to $\mathcal{S}$, and terminate.
  - If $h' = h$, then send message $(\mathsf{output}, sid, ssid, r^*)$ to $\mathcal{S}$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid, \widetilde{H}', H'')$, then send message $(\mathsf{output}, sid, ssid, r^*)$ to P_r for $r \in [n]$, and message $(\mathsf{return}, sid, ssid, \mathsf{coins}(d - q/n))$ to each P_r for $r \in H$, and message $(\mathsf{prize}, sid, ssid, \mathsf{coins}(q))$ to P_{r^*}, and message $(\mathsf{extrapay}_1, sid, ssid, \mathsf{coins}(q))$ to P_r for $r \in \widetilde{H}'$, and message $(\mathsf{extrapay}_2, sid, ssid, \mathsf{coins}(q/n))$ to P_r for $r \in H''$, and message $(\mathsf{payback}, sid, ssid, \mathsf{coins}((h - \widetilde{h}')q - h''q/n))$ to $\mathcal{S}$ where $\widetilde{h}' = |\widetilde{H}'|$ and $h'' = |H''|$, and terminate.
  - Else if $\mathcal{S}$ returns $(\mathsf{abort}, sid, ssid)$, send messages $(\mathsf{return}, sid, ssid, \mathsf{coins}(d))$ and $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q))$ to P_r for all $r \in H$, and messages $(\mathsf{sendback}, sid, ssid, \mathsf{coins}(tq/n))$ to $\mathcal{S}$, and terminate.

Fig. 3. The ideal functionality $\mathcal{F}^{\star}_{\mathrm{lot}}$ for secure lottery with penalties.

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 $\mathcal{F}^{\star}_{\text{lot}}$ for secure lottery with penalties in Figure 3. We elaborate more on the definition of the ideal functionality $\mathcal{F}^{\star}_{\text{lot}}$ below.

Ideal functionality $\mathcal{F}^{\star}_{\text{lot}}$. The high level idea behind the design of $\mathcal{F}^{\star}_{\text{lot}}$ is the same as that for $\mathcal{F}^{\star}_{f}$. 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., $\mathsf{coins}(q/n)$). Thus we require that each honest party makes a fixed lottery deposit $\mathsf{coins}(d)$ with $d \geq q/n$. Then, in the second phase, as was the case with $\mathcal{F}^{\star}_{f}$, the ideal functionality $\mathcal{F}^{\star}_{\text{lot}}$ allows $\mathcal{S}$ to learn the outcome of the lottery only if it made a sufficient penalty deposit (i.e., $\mathsf{coins}(hq + (tq/n))$). 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. $H', \widetilde{H}', H''$) in the definition of $\mathcal{F}^{\star}_{f}$ (resp. $\mathcal{F}^{\star}_{\text{lot}}$) 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 $\mathcal{F}^{\star}_{f}$ and $\mathcal{F}^{\star}_{\text{lot}}$. We leave this for future work.

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) the special ideal functionality $\mathcal{F}^{\star}_{\text{CR}}$ 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, $\mathsf{pubNMSS}$). Secure computation with penalties is then achieved via carrying out "fair reconstruction" for the $\mathsf{pubNMSS}$ scheme.[12]

First, we provide a high level description of the semantics of the $\mathsf{pubNMSS}$ scheme. The Share algorithm takes as input a secret u, and generates "tag-token" pairs $\{(\mathsf{Tag}_i, \mathsf{Token}_i)\}_{i\in[n]}$. Finally it outputs to each party P_i the i-th token Token_i and $\mathsf{AllTags} = (\mathsf{Tag}_1, \ldots, \mathsf{Tag}_n)$. Loosely speaking, the properties that we

[12] Our strategy is similar to the use of non-malleable secret sharing in [33] 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.

require from $\mathsf{pubNMSS}$ 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, $\mathsf{AllTags}$ does not reveal any further information), and (2) for any $j \in [n]$, the adversary cannot reveal $\mathsf{Token}'_j \neq \mathsf{Token}_j$ such that $(\mathsf{Tag}_j, \mathsf{Token}'_j)$ 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 $\mathsf{pubNMSS}$ scheme can be obtained via *commitments* that are binding for *honest* sender (exactly as in [26]) and are equivocal. Instantiating a variant of the Naor commitment scheme [48] as done in [26], we obtain a construction of a $\mathsf{pubNMSS}$ scheme using only *one-way functions.* (See full version [13] for more details.) We do not attempt to provide a formal definition of $\mathsf{pubNMSS}$ schemes. Rather, 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 $u_1, \ldots, u_n$.
- To generate the i-th "tag-token" pair, apply the sender algorithm for a honest-binding commitment using randomness ω_i to secret share u_i to obtain com_i, and set $\mathsf{Tag}_i = \mathsf{com}_i$ and $\mathsf{Token}_i = (u_i, \omega_i)$.

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

Next we show how to perform "fair reconstruction" for this scheme.

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.

In this section, we show how to design a protocol for fair reconstruction in the $\mathcal{F}^\star_{\mathrm{CR}}$-hybrid model. For lack of space, we refer to the full version for intuition, detailed description, and a proof of security of our protocol.

Notation. As discussed before, we assume that the secret has been shared using $\mathsf{pubNMSS}$, i.e., each party P_i now has $\mathsf{AllTags}$ and its own token 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 Token_i. A sender P_s may use (a set of) tags to specify a $\mathcal{F}^\star_{\mathrm{CR}}$ 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 $\mathcal{F}^\star_{\mathrm{CR}}$ deposit transaction made by P_1 with $\mathsf{coins}(q)$ which can be claimed by P_2 in round τ only if it produces token T, and if P_2 does not claim the transaction, then P_1 gets $\mathsf{coins}(q)$

refunded back after round τ. We use $\tau_1, \ldots, \tau_n$ 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., $\tau_1 < \tau_2$.

Multiparty fair reconstruction via the "ladder" construction. We will ask parties to make deposits in two phases. In the first phase, parties $P_1, \ldots, P_n$ simultaneously make a deposit of $\mathsf{coins}(q)$ to recipient P_n that can be claimed only if tokens $T_1, \ldots, T_n$ are produced by P_n. We call these deposits roof deposits. Then, in the second phase, each P_{s+1} makes a deposit of $\mathsf{coins}(s \cdot q)$ to recipient P_s that can be claimed only if tokens $T_1, \ldots, T_s$ are produced by P_s. These deposits are called the ladder deposits. We also force P_{s+1} to make 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 4.

Roof Deposits.

$$\begin{tikzcd}
P_1 \arrow[r, "T_1 \wedge \cdots \wedge T_n", "{q, \tau_n}"'] & P_n \\
P_2 \arrow[r, "T_1 \wedge \cdots \wedge T_n", "{q, \tau_n}"'] & P_n \\
\vdots & \\
P_{n-2} \arrow[r, "T_1 \wedge \cdots \wedge T_n", "{q, \tau_n}"'] & P_n \\
P_{n-1} \arrow[r, "T_1 \wedge \cdots \wedge T_n", "{q, \tau_n}"'] & P_n
\end{tikzcd}$$

Ladder Deposits.

$$\begin{tikzcd}
P_n \arrow[r, "T_1 \wedge \cdots \wedge T_{n-1}", "{(n-1)q, \tau_{n-1}}"'] & P_{n-1} \\
P_{n-1} \arrow[r, "T_1 \wedge \cdots \wedge T_{n-2}", "{(n-2)q, \tau_{n-2}}"'] & P_{n-2} \\
\vdots & \\
P_3 \arrow[r, "T_1 \wedge T_2", "{2q, \tau_2}"'] & P_2 \\
P_2 \arrow[r, "T_1", "{q, \tau_1}"'] & P_1
\end{tikzcd}$$

Fig. 4. Roof and Ladder deposit phases for fair reconstruction.

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).

The deposits are then 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. See full version for a formal description of the protocol in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model. Since $\mathcal{F}_{\mathrm{OT}}$, the ideal functionality for oblivious transfer, is sufficient [37, 39] to compute any standard ideal functionality 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* $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}^{\star}_{\mathrm{CR}})$*-hybrid model. Further, the protocol requires* $O(n)$ *rounds, a total of* $O(n)$ *calls to* $\mathcal{F}^{\star}_{\mathrm{CR}}$*, and each party deposits* $O(n)$ *times the penalty amount.*

Somewhat surprisingly, minor modifications to the above protocol leads us to a construction for secure lotteries with penalties.

4 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 $\mathsf{pubNMSS}$ (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 modifed 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 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 ϕ^j_{rf} by party P_j such that it pays q coins to P_n only if P_j did not win the lottery.[13] 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 2. *Assuming the existence of one-way functions, there exists a n-party protocol for* secure lottery with penalties *in the* $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}^{\star}_{\mathrm{CR}})$*-hybrid model. Further, the protocol requires* $O(n)$ *rounds, a total of* $O(n)$ *calls to* $\mathcal{F}^{\star}_{\mathrm{CR}}$*, and each party is required to deposit* $O(n)$ *times the penalty amount.*

Acknowledgments. We would like to thank Yuval Ishai for many useful discussions. The first author thanks Eli Ben-Sasson for his encouragement and support. We would also like to thank the anonymous referees of Crypto 2014 for their valuable comments.

References

1. M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Fair two-party computations via the bitcoin deposits. In *ePrint 2013/837*, 2013.
2. M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.
3. G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *Theory of Cryptography Conference*, 2014.

[13] Formally, for $s \in [n]$, define $\phi^s_{\mathsf{lad}}(T_1, \ldots, T_s) = \phi(\mathsf{Tag}_1, T_1) \wedge \cdots \wedge \phi(\mathsf{Tag}_s, T_s)$. For all $s \in [n-1]$, define $\phi^s_{\mathsf{rf}}(T_1, \ldots, T_n) = \phi^n_{\mathsf{lad}}(T_1, \ldots, T_n) \wedge (\mathsf{Ext}(\mathsf{Tag}_1, T_1) + \cdots + \mathsf{Ext}(\mathsf{Tag}_n, T_n) \neq s \bmod n)$, where Ext extracts the exact share (i.e., the input for the commitment) from token T.

4. N. Asokan, V. Shoup, and M. Waidner. Optimistic protocols for fair exchange. In *ACM CCS*, pages 7–17, 1997.
5. N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *Eurocrypt*, pages 591–606, 1998.
6. A. Back and I. Bentov. Note on fair coin toss via bitcoin. http://arxiv.org/abs/1402.3698, 2013.
7. S. Barber, X. Boyen, E. Shi, E. Uzun. Bitter to Better - How to Make Bitcoin a Better Currency. In *Financial Cryptography*, pages 399–414, 2012.
8. D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *IEEE FOCS*, pages 468–473, 1989.
9. A. Beimel, Y. Lindell, E. Omri, and I. Orlov. $1/p$-Secure multiparty computation without honest majority and the best of both worlds. In *Crypto* 2011.
10. 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.
11. M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts (extended abstract). In *ICALP*, pages 43–52, 1985.
12. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computations. In *ACM STOC*, 1988.
13. I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *ePrint 2014/129*, 2014.
14. D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.
15. C. Cachin and J. Camenisch. Optimistic fair secure computation. In *Advances in Cryptology — Crypto 2000*, volume 1880 of *LNCS*, pages 93–111. Springer, 2000.
16. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
17. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE FOCS*, pages 136–145, 2001.
18. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Eurocrypt*, pages 68–86, 2003.
19. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *ACM STOC*, pages 11–19, 1988.
20. L. Chen, C. Kudla, and K. Paterson. Concurrent signatures. In *Eurocrypt*, pages 287–305, 2004.
21. J. Clark and A. Essex. Commitcoin: Carbon dating commitments with bitcoin - (short paper). In *Financial Cryptography*, pages 390–398, 2012.
22. R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.
23. E. Friedman and P. Resnick. The social cost of cheap pseudonyms. In *Journal of Economics and Management Strategy*, pages 173–199, 2000.
24. J. Garay and M. Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography and Data Security*, pages 168–182, 2002.
25. J. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology — Crypto '99*, pages 449–466. Springer, 1999.
26. J. Garay, J. Katz, R. Kumaresan, and H-S. Zhou. Adaptively secure broadcast, revisited. In *ACM PODC*, pages 179–186, 2011.
27. J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *TCC*, pages 404–428, 2006.
28. J. Garay and C. Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In *Financial Cryptography and Data Security*, pages 190–207, 2003.

29. O. Goldreich. Foundations of cryptography - volume 2, basic applications. 2004.
30. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *ACM STOC*, pages 218–229, 1987.
31. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology — Crypto '90*, pages 77–93, 1991.
32. S. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In *ACM STOC*, pages 413–422, 2008.
33. S. Gordon, Y. Ishai, T. Moran, R. Ostrovsky, and A. Sahai. On complete primitives for fairness. In *TCC*, pages 91–108, 2010.
34. S. Gordon and J. Katz. Partial fairness in secure two-party computation. In *Eurocrypt*, pages 157–176, 2010.
35. Y. Huang, J. Katz, and D. Evans. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
36. Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, and A. Malozemoff. Amortizing garbled circuits. In *Crypto*, 2014.
37. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Crypto 2008*, pages 572–591, 2008.
38. J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous communication. In *TCC*, pages 477–498, 2013.
39. Joe Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.
40. A. Küpçü and A. Lysyanskaya. Optimistic fair exchange with multiple arbiters. In *ESORICS 2010*, pages 488–507, 2010.
41. A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. In *CT-RSA*, pages 252–267, 2010.
42. A. Lindell. Legally-enforceable fairness in secure two-party computation. In *Cryptographers' Track — RSA 2008*, pages 121–137, 2008.
43. Y. Lindell and B. Riva. Cut-and-choose yao-based two-party computation with low cost in the online/offline and batch settings. In *Crypto*, 2014.
44. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay: a secure two-party computation system. In *USENIX*, pages 20–20, 2004.
45. G. Maxwell. Zero knowledge contingent payment, 2011. `https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment`.
46. S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *ACM PODC*, pages 12–19, 2003.
47. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `http://bitcoin.org/bitcoin.pdf`.
48. M. Naor. Bit commitment using pseudo-randomness. In *Advances in Cryptology — Crypto '89* pages 128–136, 1990.
49. B. Pinkas. Fair secure two-party computation. In *Eurocrypt*, pages 87–105, 2003.
50. 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.