# Round Efficiency of Multi-Party Computation with a Dishonest Majority

Jonathan Katz[1], Rafail Ostrovsky[2], and Adam Smith[3] *

[1] Dept. of Computer Science, University of Maryland, College Park, MD. jkatz@cs.umd.edu
[2] Telcordia Technologies, Morristown, NJ. rafail@research.telcordia.com
[3] MIT Lab. for Computer Science, Cambridge, MA. adsmith@mit.edu

**Abstract.** We consider the round complexity of multi-party computation in the presence of a static adversary who controls a *majority* of the parties. Here, $n$ players wish to securely compute some functionality and up to $n-1$ of these players may be arbitrarily malicious. Previous protocols for this setting (when a broadcast channel is available) require $O(n)$ rounds. We present two protocols with improved round complexity: The first assumes only the existence of trapdoor permutations and dense cryptosystems, and achieves round complexity $O(\log n)$ based on a proof scheduling technique of Chor and Rabin [13]; the second requires a stronger hardness assumption (along with the non-black-box techniques of Barak [2]) and achieves $O(1)$ round complexity.

## 1 Introduction

Protocols for secure multi-party computation (MPC) allow a set of $n$ parties to evaluate a joint function of their inputs such that the function is evaluated *correctly* and furthermore no information about any party's input — beyond what is leaked by the output of the function — is revealed (a formal definition is given in Section 2). Since the initial results showing that MPC was feasible [34, 24, 7, 12], a number of works have focused on improving the efficiency of these protocols and in particular their round complexity (e.g., [1, 6, 29, 28, 22, 30, 15]). Known results for generic MPC secure against malicious adversaries in the computational setting may be summarized as follows (results are stated for the setting when a broadcast channel is available; we discuss the setting without a broadcast channel in Section 2.1):

- Secure two-party computation may be achieved in a constant number of rounds by applying the compiler of Lindell [30] (based on earlier work of Goldreich, Micali, and Wigderson [24]) to the constant-round protocol of Yao [34] (which is secure against semi-honest adversaries).

* Supported in part by U.S. Army Research Office Grant DAAD19-00-1-0177

- Secure MPC for honest majorities (i.e., when the number of corrupted players is strictly less than $n/2$) may be achieved in a constant number of rounds using the protocol of Beaver, Micali and Rogaway [6, 33].
- Secure MPC with dishonest majority (i.e., where up to $n-1$ players may be corrupted) can be achieved in $O(n)$ rounds using the protocols of Beaver, Goldwasser, and Levin [5, 26]. (Actually, these works show a protocol requiring $O(k+n)$ rounds where $k$ is the security parameter. Using the techniques of [30], however, this may be improved to $O(n)$.)
- Canetti, et al. [11] give a protocol tolerating *adaptive* adversaries controlling a dishonest majority in a model in which a common random string is assumed; the round complexity of this protocol depends on the depth of the circuit for the function being computed, but is independent of $n$.

Note that the setting with a dishonest majority ($t \geq n/2$) requires a weaker variant of the usual definition of MPC. Even for the case $n = 2$, one cannot prevent the adversary from aborting the protocol, or from possibly learning information about the value of the function even when an abort occurs [24, 14].

OUR RESULTS. We focus on improving the round complexity of MPC when a majority of the players may be corrupted. We show two protocols for that setting which have improved round complexity compared to previous work. Our first protocol assumes the existence of trapdoor permutations and dense cryptosystems, and achieves round complexity $O(\log n)$. Our second protocol runs in a *constant* number of rounds, but requires slightly stronger hardness assumptions as well as non-black-box proof techniques. We prove our results in the standard model of a synchronous, complete network with a broadcast channel. Our results can be extended to the setting when no broadcast channel is available, and give improved round complexity there as well; see Section 2.1.

Our overall approach consists of two steps. We first consider the specific case of the *coin flipping* functionality, and give protocols for securely computing this functionality in the presence of a dishonest majority. We then note that MPC of *arbitrary* functions can be reduced to the problem of secure coin flipping; in fact, we show that any functionality can be computed in a constant number of rounds following an execution of a secure coin-flipping protocol.

Our main result, then, is to give two protocols with improved round complexity for the specific case of coin flipping. The first, based on a proof scheduling technique of Chor and Rabin [13], requires $O(\log n)$ rounds. (Interestingly, the Chor-Rabin protocol itself does *not* seem sufficient to implement MPC; we need to first establish a common random string and then use that string for secure computation.) Our second coin-flipping protocol extends recent work of Barak [2]; specifically, we show how to modify his (asynchronous) two-party non-malleable coin-flipping protocol to obtain one which is secure even when composed in parallel $n$ times, and from there obtain a constant-round coin-flipping protocol which is secure in the (synchronous) multi-party setting. We may thus summarize our results as follows (here, $n$ is the number of players, $k$ is the security parameter, and we always assume a synchronous network with broadcast):

**Theorem 1.1.** *There exist protocols for $(n-1)$-secure simulatable coin-flipping with the following properties:*

*1. $O(\log n)$ rounds, assuming one-way permutations.*

*2. $O(1)$ rounds, assuming collision-free hashing, trapdoor permutations, and dense cryptosystems secure against $2^{k^\epsilon}$-size circuits. The proof uses a non-black-box simulation.*

**Theorem 1.2.** *For any poly-time function $f$, there exist $(n-1)$-secure protocols for computing $f$ with the following properties:*

*1. $O(\log n)$ rounds, assuming trapdoor permutations and dense cryptosystems.*

*2. $O(1)$ rounds, assuming collision-free hashing, trapdoor permutations, and dense cryptosystems secure against $2^{k^\epsilon}$-size circuits. The proof uses a non-black-box simulation.*

Note that information-theoretically secure protocols are impossible in our setting: generic MPC protocols tolerating $t \geq n/2$ imply the existence of two-party oblivious transfer protocols, which require computational assumptions [29].

In Section 2 we specify our model and definition of MPC. Section 3 shows how to achieve coin flipping in logarithmic rounds; the constant-round protocol is explained in Section 4. Section 5 shows how to achieve MPC for arbitrary functionalities given a protocol for secure coin flipping.

## 2 Definitions

Our definition of security for MPC is taken from the works of Canetti [8] and Goldwasser and Lindell [27], which in turn follow a long line of work on defining security of protocols (e.g., [24, 26, 31, 4, 23]). More recently, a stronger definition of *universally composable* (UC) computation has been proposed [9]; however, UC-MPC is known to be impossible in the presence of a dishonest majority without the prior assumption of a common random string [10]. Since we wish to avoid a setup assumption of this form (indeed, we give explicit protocols for obtaining a common random string), we do not use the UC framework directly. Nonetheless, some of the protocols we use as building blocks were proven secure in the UC framework, a fact which highlights the utility of such definitions.

### 2.1 Network Model

Formal definitions of security are given below, but we provide an overview of our model and definition of security here. We consider a system of $n$ parties who interact in a synchronous manner. Each pair of parties is connected by a perfect (authenticated, secret, unjammable) point-to-point channel, and we also assume a broadcast channel to which all players have access. This channel provides authenticity (i.e., that a given broadcast message originated from a particular party) and also ensures that all parties receive the same message even if the broadcasting party is dishonest. Messages sent on any of these channels are delivered in the same round they are sent.

We assume a static adversary who corrupts up to $n-1$ of the players before execution of the protocol. The adversary is active, and corrupted parties may behave in an arbitrary manner. Although the adversary may not delay or block messages from honest parties, we do make the standard *rushing* assumption: i.e., the adversary sees all messages sent by honest players to corrupted players at a given round $i$ (including broadcast messages) before sending its own messages for round $i$. Finally, we consider *computational* security only and therefore restrict our attention to adversaries running in probabilistic, polynomial time.

Although we assume a broadcast channel, our techniques yield protocols with improved round complexity even when broadcast is not available. When only point-to-point links are assumed, broadcast may be implemented using an $O(t)$-round authenticated Byzantine agreement protocol (where $t < n$ players are dishonest) [18]; this protocol assumes a pre-existing public-key infrastructure (PKI) but in our context a PKI may be constructed "from scratch" in $O(t)$ rounds without affecting overall security of the protocol [20] (this relies on the fact that the adversary is allowed to abort when controlling a dishonest majority). Thus, when broadcast is unavailable our techniques reduce the round complexity of known MPC protocols from $O(tn)$ to $O(t \log n)$ using our first coin-flipping protocol, or $O(t)$ using our second coin-flipping protocol. (For a weaker version of MPC in which the honest players need not agree on whether or not the protocol aborted [27], only a constant increase in round complexity is necessary over the broadcast-based protocols given here [27].) The assumption of a broadcast channel is therefore made for simplicity of exposition only.

### 2.2 Secure Multi-Party Computation and Coin-Flipping

Following the outline of [8, 27], we define an ideal model of computation and a real model of computation, and require that any adversary in the real model can be *emulated* (in the specific sense described below) by an adversary in the ideal model. Our randomized function $f$ to be computed by the $n$ parties is denoted by $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ where $f = (f_1, \ldots, f_n)$; that is, for a vector of inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$, the output is a vector of values $(f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$. Note that we may also view $f$ as a deterministic function on $n+1$ inputs, where the final input represents the random coins used in evaluating $f$. In a given execution of the protocol we assume that all inputs have length $k$, the security parameter.

IDEAL MODEL. In the ideal model there is a trusted party which computes the desired functionality based on the inputs handed to it by the players. Let $I \subset [n]$ denote the set of players corrupted by the adversary. Then an execution in the ideal model proceeds as follows [23] (this particular definition is called *secure computation with unanimous abort and no fairness* in the taxonomy of [27]):

**Inputs** Each party $i$ has input $x_i$. We represent the vector of inputs by $\boldsymbol{x}$.

**Send inputs to trusted party** Honest parties always send their inputs to the trusted party. Corrupted parties, on the other hand, may decide to abort or to send modified values to the trusted party. Let $\boldsymbol{x}'$ denote the vector of inputs received by the trusted party.

**Trusted party sends results to adversary** If $\boldsymbol{x}'$ is a valid input (i.e., no parties aborted in the previous round), the trusted party generates uniformly-distributed random coins, computes $f(\boldsymbol{x}') = (y_1, \ldots, y_n)$, and sends $y_i$ to party $P_i$ for all $i \in I$. In case a party aborted in the previous round, the trusted party sends $\perp$ to all parties.

**Trusted party sends results to honest players** The adversary, depending on its view up to this point, may decide to abort the protocol. In this case, the trusted party sends $\perp$ to the honest players. Otherwise, the trusted party sends $y_i$ to party $P_i$ for each $i \notin I$.

**Outputs** An honest party $P_i$ always outputs the response $y_i$ it received from the trusted party. Corrupted parties output $\perp$, by convention. The adversary outputs an arbitrary function of its entire view throughout the execution of the protocol.

For a given adversary $\mathcal{A}$, the *execution of $f$ in the ideal model* on input $\boldsymbol{x}$ (denoted $\text{IDEAL}_{f,\mathcal{A}}(\boldsymbol{x})$) is defined as the vector of the outputs of the parties along with the output of the adversary resulting from the process above.

REAL MODEL. As described in Section 2.1, we assume a synchronous network with rushing. Honest parties follow all instructions of the prescribed protocol, while corrupted parties are coordinated by a single adversary and may behave arbitrarily. At the conclusion of the protocol, honest parties compute their output as prescribed by the protocol, while corrupted parties output $\perp$. Without loss of generality, we assume the adversary outputs exactly its entire view of the execution of the protocol. For a given adversary $\mathcal{B}$ and protocol $\Pi$ for computing $f$, the *execution of $\Pi$ in the real model* on input $\boldsymbol{x}$ (denoted $\text{REAL}_{\Pi,\mathcal{B}}(\boldsymbol{x})$) is defined as the vector of outputs of the parties along with the output of the adversary resulting from the above process.

Having defined these models, we now define what is meant by a secure protocol. (Note: By *probabilistic polynomial time* (PPT), we mean a probabilistic Turing machine with non-uniform advice whose running time is bounded by a polynomial in the security parameter $k$. By *expected probabilistic polynomial time* (EPPT), we mean a Turing machine whose *expected* running time is bounded by some polynomial, for *all* inputs.)

**Definition 2.1 ([8]).** *Let $f$ and $\Pi$ be as above. Protocol $\Pi$ is a $t$-*secure protocol for computing $f$ *if for every* PPT *adversary $\mathcal{A}$ corrupting at most $t$ players in the real model, there exists an* EPPT *adversary $\mathcal{S}$ corrupting at most $t$ players in the ideal model, such that:*

$$\left\{\text{IDEAL}_{f,\mathcal{S}}(\boldsymbol{x})\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n} \stackrel{c}{\equiv} \left\{\text{REAL}_{\Pi,\mathcal{A}}(\boldsymbol{x})\right\}_{\boldsymbol{x}\in(\{0,1\}^*)^n} .$$

As mentioned in the Introduction, our protocols for MPC proceed in the following way: First, a common random string is generated using a coin-flipping protocol; next, the resulting string is used by the parties for the remainder of their execution. Thus, using a simple composition result, we may construct our protocols for $(n-1)$-secure MPC in two steps: (1) construct an $(n-1)$-secure coin-flipping protocol (i.e., a protocol computing the functionality $f(1^m, \ldots, 1^m) \mapsto U_m$, where $U_m$ denotes the uniform distribution over $\{0,1\}^m$); and (2) construct an $(n-1)$-secure protocol for evaluating any functionality in the common random string model (i.e., where all parties are first given a uniformly-distributed string of the appropriate length). Step (1) is discussed in Sections 3 and 4, and step (2) and the composition theorem are discussed in Section 5.

Since our main contributions are our protocols for coin flipping (achieving $(n-1)$-secure MPC in the common random string model is relatively straightforward), and since the definition of security simplifies considerably in this case, we present a stand-alone definition here. Note that the definition does *not* reduce to the most simplistic notion of coin-flipping in which we simply have a guarantee that the output of the protocol is indistinguishable from random. Instead, it must be that a simulator can produce a view which is indistinguishable from that of the real adversary, but where the outcome has been forced to be a particular random string provided by an outside party. [1] Thus, we refer to the notion as "simulatable coin flipping" (even though this is precisely the same notion as $(n-1)$-secure evaluation of the coin-flipping functionality).

**Definition 2.2 (Simulatable Coin Flipping).** *A protocol $\Pi$ is a* simulatable coin-flipping protocol *if it is an $(n-1)$-secure protocol realizing the coin-flipping functionality. That is, for every* PPT *adversary $\mathcal{A}$ corrupting at most $n-1$ parties, there is an* EPPT *machine $\mathcal{S}_\mathcal{A}$ such that the outcomes of the following experiments are computationally indistinguishable (as a function of $k$):*

| $\mathrm{REAL}(1^k, 1^m)$ | $\mathrm{IDEAL}(1^k, 1^m)$ |
|---|---|
| $c, View_\mathcal{A} \leftarrow \mathrm{REAL}_{\Pi,\mathcal{A}}(1^k, 1^m)$ | $c' \leftarrow \{0,1\}^m$ |
| | $\tilde{c}, View \leftarrow \mathcal{S}_\mathcal{A}(c', 1^k, 1^m)$ |
| OUTPUT $(c, View_\mathcal{A})$ | *If* $\tilde{c} \in \{c', \perp\}$, OUTPUT $(\tilde{c}, View)$ |
| | *Else* OUTPUT fail |

*Here we parse the result of running protocol $\Pi$ with adversary $\mathcal{A}$ (denoted $\mathrm{REAL}_{\Pi,\mathcal{A}}(1^k, 1^m)$) as a pair $(c, View_\mathcal{A})$ where $c \in \{0,1\}^m \cup \{\perp\}$ is the outcome and $View_\mathcal{A}$ is the adversary's view of the computation.*

## 3 Simulatable Coin-Flipping in $O(\log n)$ Rounds

In order to construct a simulatable coin-flipping protocol, we will use a protocol in which all pairs of players can prove statements (in zero-knowledge) to each other. More precisely, suppose that each player $P_i$ has a (publicly known) statement $x_i$ and each honest player also has private input $w_i$ (where $w_i$ is a witness for $x_i$). We would like a protocol in which each player $P_i$ proves that $x_i$ is true (and that furthermore, $P_i$ knows a witness); upon completion of this protocol, all

[1] A related notion of simulatable bit-commitment was considered in [32].

honest players should accept the result if and only if all players have successfully completed their proofs.

The naive approach to solving this problem is to have every (ordered) pair of players $P_i, P_j$ simultaneously execute some constant-round zero-knowledge proof of knowledge in which $P_i$ proves knowledge of $w_i$ to $P_j$. However, such an approach *does not work* (in general) due to the potential malleability of the proof system. Namely, it is possible that an adversary controlling $P_j$ could divert a proof being given to $P_j$ by $P_i$ and hence prove a false statement (or, at least, one for which $P_j$ does not explicitly know a witness) to $P_k$. In particular, this is *always* possible without some mechanism to prevent simple copying of proofs.

An alternate approach — one taken by previous work in the case of dishonest majority [5, 26] — is to have each pair of parties execute their proofs *sequentially* over a total of $n$ "stages" of the protocol. In stage $i$, player $P_i$ proves knowledge (in parallel) to all other players. This clearly avoids the malleability problem discussed above, but results in an $O(n)$-round protocol.

In fact, the issue of proof scheduling was previously dealt with by Chor and Rabin [13] in the context of *mutually independent commitments*. They proposed a scheduling strategy which results in a round complexity of $O(\log n)$. The scheduling guarantees that at any given time, no player is playing both the prover and the verifier. Moreover, every player eventually proves to every other player. This means that no matter what set of players is controlled by the adversary, he will eventually have to prove all his statements to some honest player. We present the Chor-Rabin (CR) scheduling strategy in Protocol 1.

To use CR scheduling in our context, we will require a zero-knowledge argument of knowledge (ZKAK) which satisfies two additional properties (informally):

*Public verifiability:* A third party who views a transcript of an execution of the proof should be able to determine in polynomial time whether or not an honest verifier would have accepted.

*Parallel composability* In our application, $n/2$ copies of the proof system will be run synchronously and in parallel. We require the existence of: (1) a simulator that can simulate the view of a dishonest verifier executing $n/2$ copies of the proof system in parallel with independent provers; and (2) a witness extractor that can extract a witness for each proof from a malicious prover who is executing $n/2$ proofs in parallel with independent verifiers.

Although not all ZKAKs satisfy both the above properties [25], the 5-round ZKAK of Feige and Shamir [19] (which only requires one-way functions) does.

Chor and Rabin [13] proved that when the $\{x_i\}$ are commitments and the $\{w_i\}$ are the corresponding decommitments, their proof-scheduling technique guarantees mutually independent commitments. However, to use the protocol as a module in a larger protocol (i.e., as in a GMW-style compiler from the honest-but-curious model to the malicious model [24]), a more sophisticated notion of security is necessary. Specifically, it is tempting to try to prove that CR scheduling realizes the ideal functionality of *mutually independent proofs*, that is, the functionality in which all players hand their pair $(x_i, w_i)$ to a trusted party who broadcasts only the list of players who supplied valid pairs.

**Protocol 1 (Chor-Rabin proof scheduling).**
Inputs: Player $i$ holds $(w_i, x_1, ..., x_n)$.
For $i = 1, .., n$, let $c_i^{(1)}, ..., c_i^{(r)}$ denote the $r = \lceil \log n \rceil$-bit binary representation of $i$.
Let $Blue_t = \left\{ i : c_i^{(t)} = 0 \right\}$ and $Red_t = \left\{ i : c_i^{(t)} = 1 \right\}$.
Let $(P, V)$ denote a constant-round publicly verifiable, parallel-composable ZKAK.

1. For $t = 1, ..., r = \lceil \log n \rceil$, repeat: $O(n^2)$ **pairs of proofs in parallel.**
   (a) $\forall i \in Blue_t, j \in Red_t$: $P_i$ runs $P(x_i, w_i)$, $P_j$ runs $V$.
   (b) (After all proofs of (a) are finished)
   $\forall i \in Blue_t, j \in Red_t$: $P_j$ runs $P(x_j, w_j)$, $P_i$ runs $V$.

*All messages are sent over the broadcast channel.* If any proof between any pair of parties fails, all players abort immediately.

It seems that the CR protocol does *not* satisfy this stronger property. Suppose the players use a malleable ZK proof system for which it is possible, given access to a prover for *either* $x_1$ *or* $x_2$, to prove knowledge of a witness for the statement $x_1 \vee x_2$. (Artificial examples of such systems can be constructed.[2]) Consider an execution of the protocol for which only $P_1$ and $P_2$ are honest. Player $P_3$ is never proving to both $P_1$ and $P_2$ simultaneously—only ever to one or the other. Moreover, when $P_3$ is proving to $P_1$, then $P_2$ is proving to some other corrupted player, and similarly when $P_3$ is proving to $P_2$. Thus, $P_3$ could claim the statement $x_1 \vee x_2$ without ever knowing an actual witness, and successfully pass all the proving stages.

Nevertheless, the CR scheduling protocol does satisfy very strong properties when the adversary controls all but one player, and this is sufficient for our purposes. The formulation of the property as it appears here is inspired by the notion of *witness-extended emulation*, due to Lindell [30].

**Lemma 3.1 (Chor-Rabin Scheduling).** *When Chor-Rabin scheduling is instantiated with any parallel-composable, publicly verifiable* ZKAK *we have:*

*Completeness: If all players are honest, and $R(x_i, w_i) = 1$ for all $i$, then all players will accept the output of the protocol.*

*Simulatability: For a machine $\mathcal{A}$, let $\mathcal{A}_{\boldsymbol{x},r}$ denote the adversary with inputs $\boldsymbol{x} = (x_1, ..., x_n)$ and random tape $r$.*

[2] Consider the standard ZK protocol for Graph Isomorphism of $(G_0, G_1)$. Prover sends $H$ and then Verifier asks for the isomorphism $H \leftrightarrow G_b$, for random $b$. The proof for $(G_0, G_1) \vee (G'_0, G'_1)$ works as follows: Prover sends $H, H'$, Verifier replies with a bit $b$, and Prover shows isomorphisms $H \leftrightarrow G_{b_1}$ and $H' \leftrightarrow G'_{b_2}$ such that $b = b_1 \oplus b_2$. A cheating intermediary who has access to a prover for $(G_0, G_1)$ or $(G'_0, G'_1)$ can fake a proof for $(G_0, G_1) \vee (G'_0, G'_1)$. A similar modification of Blum's Hamitonian Path proof system also works.

*There is a simulator $\mathcal{S}$ with inputs $1^k, \boldsymbol{x}$ and oracle access to $\mathcal{A}_{\boldsymbol{x},r}$ and two outputs: a protocol view $V$ and a list of potential witnesses $\boldsymbol{w} = (w_1, ..., w_n)$. For any* PPT *adversary $\mathcal{A}$ who controls all but one player $P_i$, $\mathcal{S}$ is* EPPT *and:*

1. *When ($\exists w_i$ s.t. $R(x_i, w_i) = 1$), the simulator's output is computationally indistinguishable from the view of $\mathcal{A}$ in an interaction with the honest $P_i$. For all $\boldsymbol{x}$: $V \stackrel{c}{\equiv} \mathsf{view}_{\mathcal{A},P_i}(\boldsymbol{x}, r)$.*
2. *When the simulated transcript is accepting, the simulator is almost certain to extract a witness for $x_j$, for all $j \neq i$:*
   $\Pr[accept_{P_i}(V) \text{ and } (\exists j \neq i: \ R(x_i, w_i) = 0)] < negl(k)$.

*Proof.* Completeness of the protocol follows directly from the completeness of the ZKAK, and so we turn to simulatability. The proof follows the reasoning of [13]. Without loss of generality, say the adversary controls all players except $P_1$. From the perspective of $P_1$, the Chor-Rabin protocol is a sequence of $2\log n$ stages, where each stage consists of $n/2$ *parallel* executions of the ZKAK. In $\log n$ of these stages, $P_1$ is acting as the prover and in $\log n$ stages $P_1$ acts as a verifier. By parallel composability of the ZKAK, we immediately see that the simulator can always simulate the view of the adversary for those stages when $P_1$ acts as a prover. By the same token, in those stages when $P_1$ acts as a verifier (and assuming that all proofs given to $P_1$ by other players are successful), $P_1$ can extract witnesses for $n/2$ of the $\{x_i\}_{i\neq 1}$. That $P_1$ in fact extracts witnesses for *all* the $\{x_i\}_{i\neq 1}$ follows from the fact that every other player acts as a prover to $P_1$ at some point in the protocol. We can combine these observations to form a simulator using the witness-extended emulation technique of Lindell [30]. □

### 3.1 From Scheduled Proofs to Simulatable Coin-Flipping

To use CR scheduling for simulatable coin-flipping, we apply a technique due to Lindell [30]. Suppose that we have a non-interactive, perfectly binding commitment scheme (these can be constructed based on one-way permutations, for example). Players first commit to individual random coins and prove knowledge of the committed values. Next, they reveal the values (not the decommitment strings) and prove correctness of their decommitments. We give the resulting construction in Protocol 2. Note that primitives weaker than ZKAKs are sufficient: we may use strong witness-indistinguishable proofs of knowledge in the first phase, and zero-knowledge proofs (of membership) in the second phase. However, using these would make the protocol and proofs more cumbersome.

**Theorem 3.1.** *Protocol 2 is a simulatable coin-flipping protocol.*

*Proof.* (sketch) The simulator is given a value $c$ and needs to simulate the view of an adversary who corrupts $n-1$ players, while also ensuring that the final output of the protocol is $c$ (we ignore for the present discussion the possibility of abort). Assume without loss of generality that the adversary corrupts all players except $P_1$. The simulator begins by following steps 1 and 2 exactly, and committing to a random value $r_1$. In step 3, the simulator may extract witnesses $\{(r_j, s_j)\}_{j\neq 1}$ by

**Protocol 2 (Simulatable coin flipping).** On input $1^k, 1^m$:

1. $\forall i,\ P_i:\ c_i \leftarrow Commit(r_i; s_i)$
2. $\forall i,\ P_i$ sends $c_i$
3. Invoke CR scheduling to show that $\forall i, \exists (r_i, s_i)$ such that $c_i = Commit(r_i; s_i)$.
4. $\forall i,\ P_i$ sends $r_i$.
5. Invoke CR scheduling to show that $\forall i, \exists s_i$ such that $c_i = Commit(r_i; s_i)$.
6. Output $c = \bigoplus_{i=1}^{n} r_i$, or $\perp$ if any proofs failed.

*All messages are sent over the broadcast channel.*

Lemma 3.1 (in this case, the simulator does not even need to be able to simulate the proofs of $P_1$ since it in fact has the necessary witness).

At this point, the simulator knows $\{r_j\}_{j \neq 1}$. It sets $r_1' = c \oplus \bigoplus_{j=2}^{n} r_j$ and sends $r_1'$ in step 4. In step 5, the simulator can simulate (false) proofs that its commitment in step 1 was indeed a commitment to $r_1'$; this follows from Lemma 3.1 (in fact, here the simulator no longer needs to extract any witnesses). These simulated proofs are computationally indistinguishable from "real" proofs, thus ensuring that the entire simulated protocol is computationally indistinguishable from an actual execution of the protocol. □

## 4 Simulatable Coin Flipping in Constant Rounds

To obtain a constant-round coin-flipping protocol, we introduce a simple notion of parallel composability for two-party non-malleable coin-flipping protocols, and show that protocols satisfying this notion can be used to achieve multi-party coin-flipping. Although the recent two-party protocol of Barak [2] does not satisfy our notion of non-malleability, we show how to extend it so that it does.

Our resulting coin-flipping protocol is described in Protocol 3. As noted above, it relies on a modification of the coin-flipping protocol of Barak [2] which is described in Section 4.1 and is denoted by $\mathsf{NMCF}(1^k, 1^m)$ for security parameter $k$ and coin-length $m$. The protocol also uses an adaptively secure, unbounded-use non-interactive zero-knowledge proof of knowledge (NIZKPK) in Steps 4 and 5. De Santis and Persiano showed how to construct these based on dense cryptosystems (PKC) and trapdoor permutations [16].[3] The use of $n$ different strings to guarantee independence of non-interactive proofs is due to Gennaro [21].

The completeness of Protocol 3 follows trivially from the completeness of the coin-flipping protocol and the NIZKPK proof system. To prove the security of the protocol, we consider the effect of a coin-flipping protocol $(A, B)$ in the

[3] In fact, one needs only weaker primitives: a strong witness-indistinguishable proof of knowledge in Step 4 and a zero-knowledge proof in Step 5. However, these distinctions make the notation of the protocol more cumbersome.

**Protocol 3 (Constant-round simulatable coin flipping).**
Let $R(c,(x,s))$ denote the relation $c = Commit(x;s)$, where $Commit$ is a perfectly binding, non-interactive commitment scheme. Suppose that for security parameter $k$, the NIZKPK system uses a CRS of length $\ell = \ell(k,m)$.

1. **Run $2\binom{n}{2}$ protocols in parallel.** For each ordered pair $(i,j) \in [n] \times [n], i \neq j$: Run coin-tossing protocol $\mathsf{NMCF}(1^k, 1^{n\ell})$ (see Lemma 4.1) to generate a string of $n\ell$ coins which will be parsed as $n$ strings $\sigma_{i,j}^{(1)}, ..., \sigma_{i,j}^{(n)} \in \{0,1\}^\ell$.
2. $P_i$: $x_i \leftarrow \{0,1\}^m$
3. $P_i$ sends $c_i = Commit(x_i, s_i)$
4. $P_i$ sends, for $j = 1, ..., n$: $\text{NIZKPK}_{\sigma_{i,j}^{(i)}}$ of $(x_i, s_i)$ such that $R(c_i,(x_i,s_i))$.
5. $P_i$ sends $x_i$ and also, for $j = 1, ...n$: $\text{NIZKPK}_{\sigma_{i,j}^{(i)}}$ that there exists $s_i$ such that $R(c_i,(x_i,s_i))$
6. Output $\bigoplus_{i=1}^n x_i$, or $\perp$ if any previous proofs or coin-flipping protocols failed.

*All messages are sent over the broadcast channel. Honest players abort immediately if any* NIZKPK *proofs fail.*

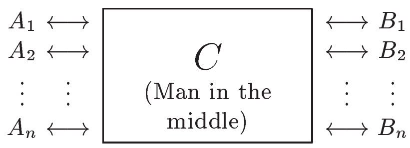


**Fig. 1.** Parallel Composition of Non-Malleable Protocols

following scenario. The adversary, $C$, simultaneously plays man-in-the-middle against with $n$ pairs of copies of the protocol executed *synchronously and in parallel* (cf. Figure 1). We call this an $n$-fold, parallel man-in-the-middle attack. It is a restriction of the more robust versions of non-malleability defined by Dolev, Dwork and Naor [17], but it seems incomparable to that of Barak [2].

Let $r_1, ..., r_n$ be the outputs of the left-hand protocols, and $\tilde{r}_1, ..., \tilde{r}_n$ be the outputs of the right-hand protocols. We clearly cannot prevent the adversary from mounting a trivial relaying attack, in which she copies messages from one or more protocols on the right to one or more protocols on the left. This allows the adversary to force the outcomes of some of the protocols to be identical. Instead, we require that the left-hand outputs $r_1, ..., r_n$ all be random and independent, and that each of the right-hand outputs $\tilde{r}_i$ is either random and independent of the others, or equal to one of the left-hand outputs $r_j$.

**Definition 4.1.** *A coin-flipping protocol $\Pi = (A, B)$ is* non-malleable against $n$-fold parallel composition *if for any* PPT *algorithm $C$, there is an* EPPT *algorithm $\hat{C}$ such that the following are computationally indistinguishable:*

1. $\mathsf{output}_{(A,B,C),\Pi}(1^k)$ *where this denotes the* $2n+1$*-tuple consisting of the* $2n$ *outputs of* $A_1, ..., A_n, B_1, ..., B_n$ *and the view of* $C$*, when executing an* $n$*-fold parallel man-in-the-middle attack.*
2. $(\rho_1, ...\rho_n, \tilde{\rho}_1, ..., \tilde{\rho}_n, \tau)$*, where first the strings* $\rho_1, ...\rho_n, \sigma_1, ...\sigma_n$ *are selected uniformly at random, and the output of* $\hat{C}(\rho_1, ...\rho_n, \sigma_1, ...\sigma_n)$ *consists of* $\tau$ *followed by a specification, for each* $i$*, of which value to assign* $\tilde{\rho}_i$ *out of* $\{\sigma_i\} \cup \{\rho_1, ..., \rho_n\}$*.*

It is not clear *a priori* that all non-malleable coin-flipping schemes satisfy this definition. In fact, it appears to be orthogonal to the definition of non-malleability in [2]: on one hand, it requires synchronous (not concurrent) execution of the $2n$ protocol pairs, and so a protocol which satisfies it may be insecure when any one of the executions is not synchronized. On the other hand, this definition requires security when the adversary has access to several protocols. In particular, if any of the building blocks of the coin-flipping protocol are not parallel-composable, then the resulting protocol may not satisfy the definition.

**Lemma 4.1.** *The coin-flipping protocol of [2] can be modified to satisfy Definition 4.1.*

We present the modified protocol in the following section. However, we first show that we can use it to obtain a constant-round simulatable coin-flipping protocol.

**Lemma 4.2.** *Protocol 3 is a simulatable coin-flipping protocol.*

*Proof.* (sketch) We begin by describing the algorithm used by the simulator, and then show that the simulation satisfies Definition 2.2. In addition to the simulator for the coin-flipping protocol, we will use the extractor and simulator for the NIZKPK system and the languages we need. The two phases of the extractor (generation and extraction) are denoted by $\mathsf{Ext}_1, \mathsf{Ext}_2$. Similarly, the simulator is denoted by $\mathsf{Sim}_1$ and $\mathsf{Sim}_2$.

On input $c \in \{0,1\}^m$, the simulator does the following:

- Pick an honest player at random (w.l.o.g. $P_1$). Allow the adversary to control the remaining honest players. That is, wrap the original adversary $\mathcal{A}$ in a circuit $\mathcal{A}'$ which makes the honest players follow the protocol. No special simulation of these players is required.
- Pick $2(n-1)$ strings $\rho_2, ..., \rho_n, \sigma_2, ..., \sigma_n$ as follows. Recall that each string is parsed as $n$ segments, each of which is long enough to serve for NIZKPK. Use the NIZKPK simulator to generate segment 1 of each string (independently), i.e. $\rho_i^{(1)}, \sigma_i^{(1)} \leftarrow \mathsf{Sim}(1^k)$ for all $i$.
  Use the NIZKPK extractor to generate segments $2, .., n$ of each string, that is $\rho_i^{(j)}, \sigma_i^{(j)} \leftarrow \mathsf{Ext}(1^k)$ for all $i$ and for $j = 2, ..., n$.
  The simulator keeps the side-information necessary for simulation and extraction with respect to each of these strings.

– (Step 1) Run the simulator $\hat{C}$ from $(n-1)$-fold parallel composition on the adversary, on inputs $\rho_2, ..., \rho_n, \sigma_2, ..., \sigma_n$. Note that here, $P_1$ is playing the roles of $A_1, .., A_{n-1}$ and $B_1, ..., B_{n-1}$. Denote the outputs of the coin flipping protocol by $\sigma_{1,j}$ and $\sigma_{j,1}$ for $j = 2, ..., n$, as in Protocol 3.
– (Steps 2, 3 and 4) Run these steps honestly: choose $x_1 \leftarrow \{0,1\}^m$, pick coins $s_1$, let $c_1 = Commit(x_1; s_1)$ and prove knowledge of $x_1$ using NIZKPK.
– Extract the values $x_2, ..., x_n$ from the proofs at Step 4 (this is possible since the values used by other players were *all* generated by the extractor for the NIZKPK). Compute $x' = c \oplus \bigoplus_{j=2}^{n} x_j$.
– (Step 5) Send $x'$. For each $j = 2, ..., n$, use the simulator for the NIZKPK to fake proofs of "$\exists s'$ such that $R(c_1, (x', s'))$" with respect to $\sigma_{1,j}^{(1)}$.
– Either the protocol aborts, or all honest players output the string $x' \oplus \bigoplus_{j=2}^{n} x_j = c$.

The proof of the success of this simulation relies on several observations. First, the strings output by the generators are pseudo-random, and so the behaviors of the adversary and simulator are the same as if the strings were truly random. By Lemma 4.1, the simulation of the NMCF protocols is indistinguishable from a real execution, and the strings generated will, with overwhelming probability, be from $\{\rho_2, ..., \rho_n, \sigma_2, ..., \sigma_n\}$.

Second, as observed by Barak [2], NIZK proof of knowledge systems remain secure even if the adversary may choose the CRS from among a polynomial set of random (or pseudo-random) strings. The adversary will not be able to make his committed values (in Step 3) dependent on those of the honest players, since that would violate the hiding property of the commitment or the zero-knowledge property of the proof system (in fact, all we need here is strong witness indistinguishability). Moreover, the simulator will be able to extract the committed values of the cheater since the adversary proves with respect to the strings generated by the extractor. Finally, the simulator's proof of consistency of his decommitment will appear legitimate, again because of the zero-knowledge property, and the adversary's proofs will have to remain sound. □

*Remark 4.1.* The use of NIZKPK in the above protocol requires a dense PKC. We expect that one can avoid this assumption by using (non-malleable) interactive ZK proofs of knowledge which rely on a public random string. We defer details to the final version.

### 4.1 Parallel Composability of Barak's Coin-Flipping Protocol

The proof of Lemma 4.1 is similar to the proofs of Theorems 2.4 and 3.4 in [2]. There are two main modifications to Barak's protocol which are necessary. First, the two proof systems that are used as sub-protocols must themselves be parallel composable. This is trivial for the strong witness-indistinguishable proof of knowledge. As for the ZK universal argument, the original paper of Barak and Goldreich [3] gives a construction which is concurrently composable and thus parallel composable.

**Protocol 4 (Parallel NM coin flipping ($\mathsf{NMCF}(1^k, 1^m)$)).**

*Steps L0.1.x, R0.1.x* (Left commits to $\alpha$): Left party chooses a hash function $h_1$, and sends $h_1$ and $y_1 = \mathsf{Com}(h_1(0^k))$. It then proves using a PSWIUAK that it knows a value $\alpha$ of length at most $k^{\log k}$ such that $y_1 = \mathsf{Com}(h_1(\alpha))$.

*Steps L0.2.x, R0.2.x* (Right commits to $\beta$): Left party chooses a hash function $h_2$, and sends $h_2$ and $y_2 = \mathsf{Com}(h_2(0^k))$. It then proves using a PSWIUAK that it knows a value $\beta$ of length at most $k^{\log k}$ such that $y_1 = \mathsf{Com}(h_1(\beta))$.

---

*Step L1* (Commitment to $r_1$): Left party selects $r_1 \leftarrow \{0,1\}^m$ and commits to it using a perfectly binding commitment scheme. The commitment is denoted $\alpha_1$.

*Steps L2.2–L2.4,R2.1–R2.3* (Prove knowledge of $r_1$): The left party proves to the right party its knowledge of the value $r_1$ committed by $\alpha_1$ using a *PSWIPOK*.

*Step R3* (Send $r_2$): The right party selects $r_2 \leftarrow \{0,1\}^m$ and sends it. *Step L4* (Send $r$) The left party sends $r = r_1 \oplus r_2$. (No decommitment string is revealed).

*Steps L5.1–5.9, R5.2–R5.10* (Prove that $r = r_1 \oplus r_2$): The left party proves, using a PZKUAK, that either $r = r_1 \oplus r_2$ or $r \in R_{\alpha\|\beta,k}$, where $\{R_{\cdot,\cdot}\}$ is an $n$-evasive set family.

Second, the evasive set family that is used in the proof of security must resist generation of an element by PPT ccircuits, even when $n$ strings from the family are given (here $n$ is the number of players and not the security parameter). By changing the union bound in the proof of existence of evasive set families ([2], Theorem 3.2), it is possible to show the existence of sets which remain evasive given $n$ elements, provided that we increase the security parameter appropriately.

The remainder of this section contains the definitions necessary to *state* the NMCF protocol (Protocol 4). The notation used in the protocol definition is taken from [2] for consistency. A proof of security is deferred to the final version. Note that here PZKUAK (resp. PSWIUAK) refers to a parallel composable *universal argument* of knowledge which is also zero-knowledge (PZKUAK) or witness-indistinguishable (PSWIUAK). These can be constructed based on trapdoor permutations and collision-free hash families secure against $2^{k^\epsilon}$-size circuits [3]. The conditions on the set family $\{R_{\cdot,\cdot}\}$ in the protocol appear below.

**Definition 4.2 ($n$-Evasive Set Family).** *Let $n = n(k) = k^c$ for some constant $c > 0$. An ensemble of sets $\{R_{\alpha,k}\}_{\alpha\in\{0,1\}^*,k\in\mathbb{N}}$, where $R_{\alpha,k} \in \{0,1\}^k$ is said to be an $n(k)$-evasive set* family *if the following conditions hold with respect to some negligible function $\mu(\cdot)$:*

*Constructibility: For any $k \in \mathbb{N}$, and any string $\alpha \in \{0,1\}^*$, the set $R_{\alpha,k}$ can be constructed in time $|\alpha|2^{k^3}$. That is, there exists a TM $M_R$ such that $M(1^k, 1^n)$ runs in time $|\alpha|2^{k^3}$ and outputs all the elements of $R_{\alpha,k}$.*

*Pseudorandomness: For all probabilistic $2^{O(k)}$-time Turing Machines $M$, and for all $\alpha \in \{0,1\}^*$, it holds that*
$\left|\Pr[r \leftarrow R_{\alpha,k} :\ M(\alpha, r) = 1] - \Pr[r \leftarrow \{0,1\}^k :\ M(\alpha, r) = 1]\right| < \mu(k)$.

*n-Evasiveness: Given* $n$ *elements of* $R_{\alpha,k}$*, it is hard for algorithms with advice* $\alpha$ *to find an* $(n+1)$*-st element: for all probabilistic* $2^{O(k)}$*-time Turing Machines* $M$*, and for any* $r_1, ..., r_n \in R_{\alpha,k}$*,* $\Pr[M(\alpha, r_1, ..., r_n) \in R_{\alpha,k} \setminus \{r_1, ..., r_n\}] < \mu(k)$.

**Definition 4.3 (String Equivalence with respect to a PRG $G$).** *Let* $G$ *be a* PRG *from* $t$ *bits to* $g(t)$ *bits secure against algorithms which take time* $o(g(t))$. *Let* $\phi(\ell)$ *be any integer function such that* $\ell < \phi(\ell) < 2^\ell$. *Consider two strings* $\alpha, \alpha' \in \{0,1\}^*$ *and let* $\ell = |\alpha| + |\alpha'|$. *The strings* $\alpha, \alpha'$ *are* $\phi$-equivalent with respect to $G$ *if there exist* $\phi(\ell)$*-time Turing machines* $M$ *and* $M'$ *which can each be described in space* $\log(\ell)$*, and such that*

$$\min \left\{ \begin{array}{c} \Pr[s \leftarrow \{0,1\}^t : M(\alpha; G(s)) = \alpha'] \ , \\ \Pr[s \leftarrow \{0,1\}^t : M'(\alpha'; G(s)) = \alpha] \end{array} \right\} > \frac{1}{\phi(\ell)}$$

*where the second input to* $M, M'$ *denotes a random tape, and* $t = g^{-1}(\phi(\ell))$.

**Lemma 4.3.** *Suppose that* $G$ *is a pseudo-random generator from* $t$ *bits to* $2^{t^\epsilon}$ *bits. Let* $\phi(\ell) = 2^{\log^{2/\epsilon}(\ell)}$. *There exists an* $n$*-evasive set family for all* $n(k) \leq k^{\epsilon/2}$*, with the additional property that if* $\alpha$ *and* $\alpha'$ *have length at most* $\ell$*, and are* $\phi$*-equivalent with respect to* $G$*, then* $R_{\alpha,k} = R_{\alpha',k}$ *for all* $k > 2^{\sqrt{\log \ell}}$.

**Proposition 4.1.** *Suppose that* $2^{k^\epsilon}$*-strong trapdoor permutations and hash families exist and that* $\{R_{\alpha,k}\}$ *is an* $n$*-evasive set family as in Lemma 4.3. Then* NMCF *(Protocol 4) is non-malleable against* $n$*-fold parallel composition.*

## 5 Multi-Party Computation

In this section we show how to obtain MPC protocols for arbitrary functionalities using any simulatable coin-flipping protocol. Let a *fixed-round protocol* be one which always requires the same number of rounds in every execution; we only discuss fixed-round protocols for poly-time computable functions $f$. Beaver, Micali and Rogaway [6] (with further extensions in [33]) shows that:

**Theorem 5.1 ([6, 33]).** *Suppose that trapdoor permutations exist. For any function* $f$*, there is an* $O(1)$*-round protocol for computing* $f$ *which is* $(n-1)$*-secure against* **honest-but-curious** *adversaries.*

For malicious adversaries, Canetti, et al. [11] construct MPC protocols in the common random string (CRS) model which are $(n-1)$-secure against *adaptive* adversaries (in fact, their protocols achieve the stronger notion of universal composability). Because their goal is security against adaptive adversaries, the round complexity of their protocols is proportional to the depth of the circuit being evaluated. Nonetheless, many of the tools they develop (such as UC commitment and zero-knowledge proofs) run in constant rounds. The following result for the case of *static* adversaries is not explicit in [11], but follows directly from their work (we use the expression *abortable* MPC to emphasize that in our setting the adversary may abort the protocol):

**Theorem 5.2 ([11]).** *Given an $r$-round protocol for* MPC *of a function $f$ which is $(n-1)$-secure against static, honest-but-curious adversaries, there is an abortable* MPC *protocol for $f$ with $O(r)$ rounds which is $(n-1)$-secure against static, malicious adversaries in the common random string model, assuming the existence of trapdoor permutations and dense* PKC.[4]

Combining the two previous theorems, we obtain:

**Corollary 5.1.** *Suppose that trapdoor permutations and dense* PKC *exist. For any function $f$, there is an $O(1)$-round (abortable) protocol for computing $f$ in the* CRS *model which is $(n-1)$-secure against static, malicious adversaries.*

The key to using simulatable coin-flipping protocols in our setting — when no setup assumptions are made and a CRS is unavailable — is the following composition result:

**Proposition 5.1.** *Given a simulatable coin-flipping protocol $\rho$, and an abortable protocol $\pi$ for computing $f$ in the* CRS *model which is $(n-1)$-secure against static, malicious adversaries, the natural composition of the two is a protocol for computing $f$* with no setup assumptions *which is $(n-1)$-secure against static, malicious adversaries.*

Canetti [8] proved a much more general composition result of this sort for the case of *non-abortable* MPC protocols. In fact, however, his proof applies in our context more or less directly. Since our particular composition result is considerably simpler, we provide a proof sketch here.

*Proof.* (sketch) Let $\mathsf{state}_{\mathcal{A}}$ denote the internal view of $\mathcal{A}$ at the end of the round in which the coin-flipping protocol $\rho$ terminates (call this round $r$). We may imagine the adversary $\mathcal{A}$ as the composition of two adversaries: $\mathcal{A}_1$ operates for $r$ rounds and produces output $\mathsf{state}_{\mathcal{A}}$. $\mathcal{A}_2$ takes as input $\mathsf{state}_{\mathcal{A}}$, operates for the remainder of the protocol and produces the final view $\mathsf{view}_{\mathcal{A}}$. We can now invoke the security of the coin-flipping protocol $\rho$ to create a simulator $\mathcal{S}_1$ which takes a string $\sigma \in \{0,1\}^m$ as input and outputs variables $\sigma', \mathsf{state}'_{\mathcal{A}}$ such that $\sigma \in \{\sigma, \perp\}$ (with overwhelming probability) and $\mathsf{state}'_{\mathcal{A}} \stackrel{c}{\equiv} \mathsf{state}_{\mathcal{A}}$ when $\sigma$ is indistinguishable from random.

We may now define an adversary $\mathcal{A}'_2$ for the CRS model as follows: upon receiving $\sigma$ from the trusted party, run $\mathcal{S}_1$ to produce $\sigma', \mathsf{state}'_{\mathcal{A}}$. If $\sigma' = \perp$, then broadcast "I abort" and halt. Otherwise, run $\mathcal{A}_2$ on input $\mathsf{state}'_{\mathcal{A}}$ to complete the protocol. Note that an execution of $\mathcal{A}'_2$ in the ideal model can be modified to yield a view and protocol outputs which are indistinguishable from those generated by $\mathcal{A}$ in the real model.[5] Finally, we invoke the security of $\pi$ to obtain a simulator $\mathcal{S}_2$ for the ideal model which emulates the behavior of $\mathcal{A}'_2$. The output of the simulator $\mathcal{S}_2$ can be similarly modified to yield outputs indistinguishable from those of $\mathcal{A}$ in the real model. □

[4] As in Remark 4.1, one should be able to remove the assumption of a dense PKC.

[5] The only difference is the "abort" message, which can simply be stripped from the transcript.

Our main result (Theorem 1.2) follows from Corollary 5.1, Proposition 5.1, and the simulatable coin-flipping protocols given in Sections 3 and 4.

## Acknowledgments

We are very grateful for helpful discussions with Cynthia Dwork, Shafi Goldwasser, Yehuda Lindell, and Moni Naor, and also for the comments from our anonymous referees. We also thank Boaz Barak for personal communication clarifying the definitions and proofs of security in [2].

## References

1. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Eighth ACM Symposium on Principles of Distributed Computing*, pages 201–209, 1989.
2. B. Barak. Constant-round coin-tossing with a man in the middle. In *43rd IEEE Symposium on the Foundations of Computer Science*, 2002. References are to the preliminary full version, available from the author's web page.
3. B. Barak and O. Goldreich. Universal arguments of knowledge. In *17th IEEE Conference on Computational Complexity*, pages 194–203, 2002.
4. D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. IACR, Springer-Verlag, Aug. 1991.
5. D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 589–590. IACR, Springer-Verlag, Aug. 1989.
6. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *22nd ACM Symposium on the Theory of Computing*, pages 503–513, 1990.
7. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th ACM Symposium on the Theory of Computing*, pages 1–10, May 1988.
8. R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd IEEE Symposium on the Foundations of Computer Science*, pages 136–147, Las Vegas, Nevada, Oct. 2001. IEEE.
10. R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. IACR, Springer, 2001.
11. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM Symposium on the Theory of Computing*, pages 494–503, Montréal, Québec, May 2002. ACM.
12. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *20th ACM Symposium on the Theory of Computing*, May 1988.
13. B. Chor and M. Rabin. Achieving independence in logarithmic number of rounds. In *6th ACM Symposium on Principles of Distributed Computing*, 1987.
14. R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *18th ACM Symposium on the Theory of Computing*, pages 364–369, 1986.

15. R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*. IACR, Springer, 2001.
16. A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *33rd IEEE Symposium on the Foundations of Computer Science*, pages 427–436. IEEE, 1992.
17. D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Computing*, 30(2):391–437, 2000.
18. D. Dolev and H. Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Computing*, 12(4):656–666, 1983.
19. U. Feige and A. Shamir. Zero knowledge proofs of knowledge in two rounds. In *Advances in Cryptology — CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 526–544. IACR, Springer-Verlag, Aug. 1989.
20. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable Byzantine agreement secure against faulty majorities. In *21st ACM Symposium on Principles of Distributed Computing*, pages 118–126, 2002.
21. R. Gennaro. Achieving independence efficiently and securely. In *ACM Symposium on Principles of Distributed Computing*, pages 130–136, 1995.
22. R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM Symposium on the Theory of Computing*, June 2001.
23. O. Goldreich. Secure multi-party computation. Electronic working draft, 2001.
24. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th ACM Symposium on the Theory of Computing*, pages 218–229. ACM, May 1987.
25. O. Goldreich and Y. Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.
26. S. Goldwasser and L. A. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology — CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer-Verlag, Aug. 1990.
27. S. Goldwasser and Y. Lindell. Secure computation without a broadcast channel. In *16th International Symposium on Distributed Computing (DISC)*, 2002.
28. Y. Ishai and E. Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41nd IEEE Symposium on the Foundations of Computer Science*, Redondo Beach, CA, Nov. 2000. IEEE.
29. J. Kilian, E. Kushilevitz, S. Micali, and R. Ostrovsky. Reducibility and completeness in private computations. *SIAM J. Computing*, 29(4), 2000.
30. Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 171–189. IACR, Springer, 2001.
31. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology — CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. IACR, Springer-Verlag, Aug. 1991.
32. M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Perfect zero-knowledge arguments for np using any one-way permutation. *J. Cryptology*, 11(2), 1998.
33. P. Rogaway. *The Round Complexity of Secure Protocols*. PhD thesis, MIT, 1991.
34. A. C.-C. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on the Foundations of Computer Science*, pages 162–167, 1986.