Pseudonymous Secure Computation from Time-Lock Puzzles

Jonathan Katz[*†] Andrew Miller[*] Elaine Shi[*‡]

Abstract

In standard models of secure computation, point-to-point channels between parties are assumed to be authenticated by some pre-existing means. In other cases, even stronger pre-existing setup—e.g., a public-key infrastructure (PKI)—is assumed. These assumptions are too strong for open, peer-to-peer networks, where parties do not necessarily have any prior relationships and can come and go as they please. Nevertheless, these assumptions are made due to the prevailing belief that nothing "interesting" can be achieved without them.

Taking inspiration from Bitcoin, we show that precise bounds on computational power can be used in place of pre-existing setup to achieve weaker (but nontrivial) notions of security. Specifically, under the assumptions that digital signatures exist and each party can solve cryptographic "time-lock" puzzles only at a bounded rate, we show that *without prior setup* and *with no bound on the number of corruptions*, a group of parties can agree on a PKI with which they can then realize *pseudonymous* notions of authenticated communication, broadcast, and secure computation. Roughly, "pseudonymous" here means that inputs/outputs are (effectively) bound to pseudonyms rather than parties' true identities.

1 Introduction

Standard models of secure computation assume that the point-to-point channels between parties are authenticated by pre-existing means, whether via some physical property of the underlying network or using keys shared by the parties in advance. When security against an unbounded number of corruptions is desired, even stronger pre-existing setup—specifically, a public-key infrastructure (PKI) or some other means of implementing broadcast—is frequently assumed.

Such setup may not exist in many interesting scenarios, especially open, peer-to-peer networks in which parties do not necessarily have any prior relationships, and can come and go as they please. Nevertheless, such setup is often assumed due to the prevailing belief that nothing "interesting" can be achieved without them. For example, Barak et al. [5] show that without authenticated channels, an adversary can partition the set of honest parties and, for each set, substitute arbitrary inputs for honest parties *outside* that set. As another example [25, 33], broadcast cannot be achieved if 1/3 or more of the participating parties can be corrupted, even given authenticated channels, without some stronger form of setup such as a PKI [12].

[*]Dept. of Computer Science, University of Maryland. **Email:** {jkatz,amiller,elaine}@cs.umd.edu.

[†]Work supported in part by NSF awards #0964541, #1111599, and #1223623.

[‡]Work supported in part by NSF award #1314857, a Sloan Research Fellowship, and a Google Faculty Research Award.

We revisit the above and ask: can nontrivial notions of security be realized without prior means of authentication or other forms of setup? Taking inspiration from Bitcoin [31] we show that, indeed, nontrivial security can be realized without setup but based instead on some assumption about the distribution of computational resources among parties. Especially, we consider a network that provides guaranteed, bounded-delay delivery of honest parties' messages, but no authentication. We also assume the existence of *time-lock* cryptographic puzzles that can be solved only at some bounded rate, plus secure digital signatures. We show that these assumptions can be leveraged to allow a set of n parties to realize *Interactive Set Consistency* (ISC), a relaxation of Interactive Consistency [33], *regardless of the number of corrupted parties.* In ISC each party has an input, and outputs a set of size n such that (1) all honest parties agree on the same set S, and (2) all honest parties' inputs are included in S. Such a protocol can naturally be used to establish a PKI among the parties; this PKI can then be used to provide "authenticated communication," "broadcast," and "secure computation with unanimous abort" [20] in the presence of an unbounded number of corruptions. (If an honest majority is assumed, we can achieve "secure computation with fairness and guaranteed output delivery.") The preceding terms are in quotes because we do not achieve them in the usual sense, but instead realize *pseudonymous* versions of them. Roughly, this means that inputs/outputs are bound to pseudonyms (effectively, parties' public keys) rather than parties' true identities.[1] We refer the reader to later sections of our paper for a formal model of the puzzles we assume, as well as a more precise definition of "pseudonymous."

Inspiration from Bitcoin. Our model is inspired by Bitcoin [31], which is widely regarded as the first broadly successful e-cash system. In place of trusted setup assumptions, Bitcoin relies (at least informally) on cryptographic puzzles and the assumption that no single entity controls more than half the computational resources in the system. The phenomenal success of Bitcoin calls for a deeper theoretical understanding of precisely what security guarantees can be achieved by these types of assumptions.

We stress, though, that our work is *not* intended to model the Bitcoin protocol itself, and we readily admit that in our initial attempt to provide a formal treatment we have made some simplifying assumptions. For example, we assume that the network guarantees (bounded-delay) delivery of messages to every party, that the number of parties is fixed (and known) at the outset, and that each party has equal computational power. Relaxing these assumptions is left for future work. Regardless, our initial results are promising and show that distributed computation models based on assumptions about resources rather than authenticated identities can lead to non-trivial security guarantees.

Related work. Okun [32] studied distributed computing problems for several variations of anonymous network models. The weakest model he considered, the anonymous "port-unaware" model, is most similar to ours, since a process does not learn whether two messages received originate from the same sender. However, our model is slightly weaker still: in the port-unaware model, each corrupt process can send at most one message to a correct process in a round, whereas in ours the adversary can deliver an arbitrary number messages to correct processes. Okun's positive result

[1] The notion of pseudonymity we mean can perhaps best be illustrated with regard to authenticated communication. The typical notion of authenticated communication is that Alice can be assured that some message she receives originated at Bob, and not some other party. *Pseudonymous* authenticated communication, in contrast, allows Alice to be assured that she receives a message from someone who calls himself "Bob," and that this is the same entity (or at least is controlled by the same entity) who sent her a message previously using the same name. However, this party may not be named Bob in any absolute sense.

here (a randomized port-unaware agreement algorithm) crucially relies on this and is inapplicable to our model - hence our use of additional assumptions about puzzles to obtain positive results. Similarly our communication model is related to (but weaker than) other models such as partial broadcast [10] and homonymous networks [11].

Cryptographic puzzles (also called *proofs of work*) have been analyzed and proposed for a variety of other purposes, including timed-release encryption [34], spam prevention [4, 13, 26], DoS resistance [21, 23], and defense against Sybil attacks [7].

Aspnes et al. [3] studied Byzantine agreement without a PKI in a model where computational puzzles can be solved at some bounded rate. This work also assumes pre-existing authenticated channels between honest parties. Moreover, their feasibility results do not extend to an unbounded number of corruptions.

The success of Bitcoin [31] has renewed interest in the use of proof-of-work puzzles for coordinating public peer-to-peer networks. For example, both [30] and [18] have presented formal models in which the core Bitcoin protocol can be shown to satisfy relaxed notions of the consensus problem given the assumption that a majority of the network is *honest*, building on the informal argument originally made in [31]. We reiterate that the goal of our work is not to analyze the Bitcoin protocol itself, but rather to explore the possibilities of related (and more abstract) models.

Our positive general results for pseudnoymous secure computation are most closely related to the prior work of Barak et al. [5], who show that a weak form of secure computation can be realized without authenticated channels. Roughly speaking, secure computation in their setting means that an adversary can partition the honest parties into disjoint subsets and then run the protocol with each of those subsets, in each case substituting inputs of its choice for the inputs of the other honest parties. The guarantees we provide here are stronger, albeit under a stronger assumption. Specifically, our notion of pseudonymous secure computation ensures that all honest parties' inputs are incorporated into a *single* computation; moreover, it ensures unanimous abort and, in the case of an honest majority, guaranteed output delivery. We also show how to achieve pseudonymous broadcast (i.e., guaranteed output delivery in this specific case) even with an unbounded number of corruptions.

Secure computation without a PKI, but when 1/3 or more of the parties can be corrupted, is also studied in [15, 16, 20]. Some of these works also achieve secure computation with unanimous abort, but none realize broadcast (with guaranteed output delivery) in the sense we do here. More importantly, all these works assume pre-existing authenticated channels between parties.

Summary of contributions and organization of the paper. In Section 2 we describe our model in more detail, as well as our formalizations of time-lock puzzles. We abstract away the details of the underlying cryptographic puzzles by introducing ideal functionalities capturing the assumption that puzzles can only be solved at a bounded rate. We consider two ideal functionalities intended to model puzzles given different interpretations of the adversary's strength f. The first functionality, $\mathcal{F}_{puz}$, models the use of proof-of-*sequential*-work puzzles (also referred to as *time-lock puzzles*) based on the assumption that the adversary's computing resources (i.e., the f corrupted processes) are *inherently parallel* and can not be used to solve a single puzzle in any faster time. The second functionality, $\mathcal{F}_{parpuz}$, is weaker, and models the use of (possibly parallelizable) proof-of-work puzzles given an adversary who can compute sequentially at a rate f times faster than a single process.

In Section 3, we define the problem of Interactive Set Consistency (ISC) and show that in the $\mathcal{F}_{puz}$-hybrid model, there is a protocol that realizes ISC against any $f < n$ corrupted parties using only $f+1$ communication rounds. We further show that this round complexity is *optimal* in this model. Additionally, we show that an $O(f^2)$-round variation of our protocol can realizes ISC in the $\mathcal{F}_{parpuz}$-hybrid model. Finally, in Section 4 we define the notion of pseudonymous secure computation and show that we can achieve it for all polynomial-time functions in the $\mathcal{F}_{puz}$-hybrid (or $\mathcal{F}_{parpuz}$-hybrid) model.

2 Our Model

Our basic underlying model is that of (stand-alone) secure computation as defined by Canetti [8]. However, we stress that in contrast to his work, we do not assume authenticated channels between the parties. We will instead work in a hybrid world where the parties can access a functionality that models their ability to solve a bounded number (here, assumed for simplicity to be one) of computational puzzles per round. We provide further details in what follows.

Network model. Our underlying model consists of n parties in a fully connected, synchronous network. Since we do not assume authenticated channels, however, some aspects of the model bear explanation. The fact that there are no authenticated channels means that when a party receives a message, it cannot tell from which other party that message originated, or whether a message it receives in one round is from the same party as some other message it received in a different round. The only thing we assume is that any message sent by an honest party in some round is received by all other parties at the end of that round. (Honest parties cannot send a message to only one specific receiver, since we do not wish to assume that parties have any prior knowledge of each others' identities. The fact that an honest party's message is *diffused* throughout the entire network is reminiscent of flooding protocols as well as how Bitcoin works.)

We consider an adversary who corrupts some parties and can cause them to behave arbitrarily. The adversary can also inject messages into the network, meaning (in particular) that an honest party may receive more than $n-1$ messages in any given round. In contrast to honest parties, we allow the adversary to send a message to any desired subset of the honest parties. We assume a *rushing* adversary who receives the messages from the honest parties in the current round before deciding on its own messages for that round. The only limitation we place on the adversary, as noted implicitly earlier, is that it may not drop or modify honest parties' messages. Thus, if an honest party sends a message in some round, then each honest party receives that message (along with whatever other messages the adversary chooses to send) at the end of that round.

The time-lock puzzle functionality, $\mathcal{F}_{puz}$. We wish to model the existence of computational puzzles that are cheap to verify but expensive (and time-consuming) to solve and, in particular, can only be solved at a bounded rate. A puzzle should also be tied to a specific value which is presented along with a solution in order to enable verification. Puzzle solutions are assumed to be uniform random strings for simplicity, though we only need them to be unpredictable.

We model the above requirements by working in a hybrid world where there is a reactive functionality $\mathcal{F}_{puz}$ which all parties can access twice per round. Let λ denote a (statistical) security parameter. The functionality maintains a set T of puzzle/solution pairs (x, h), with $x \in \{0,1\}^*$ and $h \in \{0,1\}^\lambda$; the set T is initially empty. Then, in each round, the functionality does as follows:

1. Receive from each party P_i an input $(\texttt{solve}, x^{(i)})$. (Note that only a *single* value is allowed.) For $i = 1, \ldots, n$, first check if a pair $(x^{(i)}, h^{(i)})$ has been stored in T, and if so return $h^{(i)}$ to P_i; otherwise, choose uniform $h^{(i)} \in \{0,1\}^\lambda$, return $h^{(i)}$ to P_i, and store $(x^{(i)}, h^{(i)})$ in T.

2. Receive from each party P_i an arbitrary-length vector $(\texttt{check}, (x_1^{(i)}, h_1^{(i)}), \ldots)$. Return to each party P_i the vector of values $(b_1^{(i)}, \ldots)$ where $b_j^{(i)} = 1$ iff $(x_j^{(i)}, h_j^{(i)}) \in T$.

We stress that, as in [8], we continue to assume synchrony in the hybrid world. This means that if a corrupted party does not provide input to $\mathcal{F}_{puz}$ by the end of the next "clock tick," the functionality still returns output to those parties who did provide input.

Note that this model directly enforces the characteristic "time-lock " property by requiring all the processes to submit their `solve` requests before receiving any of the puzzle solutions. While an adversary who corrupts f processes can solve f puzzles per round in total, it cannot solve one puzzle in $1/f$ of a round (i.e., a puzzle *request* made in a round cannot depend on any puzzle *solution* computed in the same round).

In detail, the precise order of events for executing each round is as follows:

1. Each party sends (at most) one puzzle-solving request to $\mathcal{F}_{puz}$ and receives the corresponding solution.
2. Each party determines a message to send.
3. Messages are delivered to each party.
4. Each party sends a list of puzzle solutions to $\mathcal{F}_{puz}$ for verification.

Instantiation of time-lock puzzles. As described above, an essential characteristic of the puzzles we model is that they cannot be solved any *faster* by the adversary than by a single honest process, even though the adversary can corrupt many processes and therefore solve *more* puzzle instances in total.

The approach taken by time-lock puzzle constructions is to have solving the puzzle require an inherently *sequential* computation, and hence an adversary with additional *parallel* computing resources cannot gain an advantage. In the first time-lock puzzle constructions [34], generating a puzzle instance required knowing the (secret) puzzle solution ahead of time; this is appropriate for applications like timed-release encryption, but unsuitable for our setting where there are no trusted parties available. A recent construction by Mahmoody et al. [28] is the first to achieve *public coin* time-lock puzzles, which do not require any trusted generation procedure. This construction is provably secure in the random-oracle model, where each oracle call is assumed to have an intrinsic sequential cost.

Given the construction mentioned above, the reader may ask why we do not work directly in the RO model. The reason is that wish to simplify the functionality provided by these puzzles, by assuming that (1) honest parties solve exactly one puzzle per round, and (2) corrupted parties solve at most one puzzle per round, and (3) checking puzzle solutions incurs no cost. The actual puzzle construction we referred to achieves this only approximately: the adversary may gain up to some constant factor of (parallel) speed-up (with better-than-negligible probability of success) and verifying a solution requires a nonzero number oracle queries (a polylogarithmic function of the number needed to produce the solution).

Relation to other notions of puzzles. The time-lock puzzles modeled by $\mathcal{F}_{puz}$ are one form of proof-of-work puzzle; however, several other notions have been discussed in the literature. Here we discuss a two alternatives and how they relate to our model.

(Possibly parallelizable) proof-of-work puzzles. All proof-of-work puzzle schemes are parameterized by a difficulty value N which indicates the expected amount of *work* needed to solve the puzzle, but the time may be reduced with parallel resources. For example, a construction due to Coelho [9] provides a very precise work guarantee – the puzzle takes almost exactly N steps to solve – although most of the steps can be taken in parallel. In the appendix, we present an variation of our model for proof-of-work puzzles generally, based on an ideal functionality $\mathcal{F}_{parpuz}$; we also present an adaptation of our main protocol for this model. (Alternatively, this can be seen as modeling time-lock puzzle constructions used in a setting where the adversary runs *sequentially* faster than honest processes by a factor of f.)

Scratch-off Puzzles. The cryptographic puzzles used in Bitcoin are not only parallelizable, but in fact can be solved by any number of concurrent processes without communication. Each process has a chance of independently producing a puzzle solution proportional to its computational power. Hence, regardless of the distribution of computing resources among honest participants in the Bitcoin network, puzzle solutions arrive according to a Poisson process. Miller et al. [30] point out that this property is essential to the operation of Bitcoin, since it guarantees that independent participants do not duplicate much work; in [29], it is argued that this process is integral to Bitcoin's incentive structure, since it ensures even weak participants have a proportional chance of finding the next puzzle solution and thereby earning a reward.

Further remarks on our simplifying assumptions. We have aimed for the simplest model that captures the features of interest. Yet the model (and our results) can be easily adapted or generalized. The assumption that the communication rate is equal to the puzzle-solving rate (i.e., that exactly one puzzle can be solved by each party in each communication round) is without much loss of generality, since the puzzle difficulty can be adjusted so this is the case. Alternately, as long as the number of puzzles that can be solved per round is bounded, our protocols can be easily modified so that they continue to provide the claimed guarantees.

Seemingly more worrisome is the assumption that the adversary has the same computational abilities as the honest parties, and so solves puzzles at the same rate. This, too, is not an essential feature, and we could easily adapt our protocols to handle an adversary who has β times more computing power than honest parties. Such an adversary can be emulated in our current model by simply viewing parties as "units of processing power," and modeling a real-world network with n parties, f of whom are corrupted, as a network with $(n - f) + \beta \cdot f$ parties, $\beta \cdot f$ of whom are corrupted. However, a distinction must be made between an adversary who has β times more *parallel* computing power (i.e., *more* processors) versus an adversary who has β times more *sequential* computing power (i.e., *faster* processors). In the former case, either the $\mathcal{F}_{puz}$ or $\mathcal{F}_{parpuz}$ model could be modified as described above; in the latter case, we would need to resort to the $\mathcal{F}_{parpuz}$ model.

We explicitly make puzzle verification "free" in our model to prevent the denial-of-service attack in which the adversary overwhelms an honest party with (incorrect) puzzle solutions. (Recall that in our model, an attacker can send as many messages as it likes to honest parties.) An alternate way to deal with such attacks would be to change our network model and assume some fixed bound on the number of messages the adversary can send honest parties.

3 Interactive Set Consistency in the $\mathcal{F}_{puz}$- Hybrid Model

We define Interactive Set Consistency (ISC), and then show an ISC protocol in the $\mathcal{F}_{puz}$-hybrid model.

Definition 1. *A protocol for n parties, in which each party P_i begins with an input value v_i, realizes* Interactive Set Consistency in the presence of f corrupted parties *if the following holds with all but negligible probability (in an associated security parameter) in the presence of any adversary controlling up to f parties:*

Boundedness *Each honest party P_i outputs a (multi)set V_i containing at most n values.*

Agreement *Each honest party P_i outputs the same (multi)set V.*

Validity *For each honest party P_i, it holds that $v_i \in V_i$.*

The ISC problem is related to Interactive Consistency (IC) [33], with the difference being that the latter has a stronger validity requirement: all honest parties agree on a *vector* $\vec{V}$ (rather than a set), and for each honest party P_i it holds that $\vec{V}[i] = v_i$. ISC can be viewed as a pseudonymous version of IC, since the honest parties may not learn which other party corresponds to each element in the set V.

We now describe a protocol for realizing ISC in the $\mathcal{F}_{puz}$-hybrid model. (In Appendix B we describe a modification of this protocol for a weaker model.) First we introduce some useful notation.

Definition 2. *A* puzzle graph *g is a tuple* $(\mathsf{sol}, \mathsf{pk}, \mathsf{children})$*, where* $\mathsf{sol} \in \{0,1\}^\lambda$ *is a puzzle solution,* pk *is an identity string, and* $\mathsf{children}$ *is a (possibly empty) set of puzzle graphs. A puzzle graph g is recursively defined to* valid *if* sol *is a solution for a puzzle with identifier* $\mathsf{puz} := \mathsf{pk}\|\mathsf{children}$*, and either* $\mathsf{children} = \emptyset$ *or else every graph in* $\mathsf{children}$ *is valid.*

The notations $\mathsf{sol}(g)$, $\mathsf{pk}(g)$, and $\mathsf{children}(g)$ are used to select components of puzzle graph g.

Definition 3. *We say that a puzzle graph h is at depth ℓ in puzzle graph g, where ℓ is a positive integer, according to the following inductive rules:*

- *Any puzzle graph g is at depth 1 in itself.*
- *A puzzle graph h is at depth $(\ell + 1)$ in g if for some $c \in \mathsf{children}(g)$, h is at depth ℓ in c.*

For short, we say h *is depth-ℓ* if it is at depth ℓ in some puzzle graph g that is clear in context.

Signed messages. In our protocols, every correct process P_i first generates signing keys $(\mathsf{pk}_i, \mathsf{sk}_i)$, and then uses sk_i to authenticate messages it sends. For convenience we always transmit the public key along with the corresponding message and signature. Since each correct process discards invalid messages upon receipt, we hereafter assume that only valid messages are sent.

Definition 4. *A* signed message *is a tuple* $(\mathsf{pk}, \sigma, \mathsf{msg})$ *where σ is a valid signature on* msg *under public key* pk.

As before, we abbreviate extraction of components of signed message s with $\mathsf{pk}(s)$, $\sigma(s)$, $\mathsf{msg}(s)$.

An $(f+1)$-Round ISC Protocol using Time-Lock Puzzles

Initially, each process P_i generates a keypair $(\mathsf{sk}_i, \mathsf{pk}'_i)$ for a digital signature scheme. For convenience, we attach the input value to the public key and let $\mathsf{pk}_i = \mathsf{pk}'_i \| v_i$, where v_i denotes the input of P_i. We refer to pk_i as the identity of process P_i.

For notational convenience, the subroutine $solve(\mathsf{pk}, \mathsf{children})$ (where $\mathsf{children}$ is a set of valid puzzle graphs) produces a new puzzle graph $g := (\mathsf{sol}, \mathsf{pk}, \mathsf{children})$, where sol is the solution returned from querying the $\mathcal{F}_{puz}$ functionality with $(\texttt{solve}, \mathsf{pk} \| \{\mathsf{sol}(c) \| c \in \mathsf{children}\})$.

Process P_i uses the subroutine $\mathsf{Sign}_i(\mathsf{msg})$ to produce a signed message $(\mathsf{pk}_i, \sigma, \mathsf{msg})$ where σ is a signature created using secret key sk_i.

- **Round 1.** In the first round, each process P_i computes the puzzle graph $g_{i,1} = solve(\mathsf{pk}_i, \emptyset)$. It then sends $g_{i,1}$ and $\mathsf{Sign}_{\mathsf{sk}_i}(\mathsf{pk}_i)$ to all parties, and sets $accepted := \{\mathsf{pk}_i\}$.
- **Round 2 through $f+1$.** In each round $2 \leq r \leq f+1$, process P_i does:
  1. Sets $G_{new} := \emptyset$, $S_{new} := \emptyset$.
  2. Receives (from the end of round $r-1$) a set of valid puzzle graphs G. Invalid puzzle graphs are discarded.
  3. Receives (from the end of round $r-1$) a set of valid signed messages S, such that $\forall s \in S$, $\mathsf{pk}(s) \in accepted$. Invalid or redundant signatures are discarded.
  4. For each $g \in G$, and for each puzzle graph h that is depth r in g, let $S_h := \{s \in S \mid \mathsf{pk}(s) = \mathsf{pk}(h)\}$. if $\mathsf{pk}(h) \notin accepted$ and $|S_h| \geq r-1$, then P_i:
     (a) sets $accepted := accepted \cup \{\mathsf{pk}(h)\}$,
     (b) sets $G_{new} := G_{new} \cup g$,
     (c) sets $S_{new} := S_{new} \cup S_h \cup \{sign_i(\mathsf{pk}(h))\}$.
  5. After processing each received message, publishes $g_{i,r} := solve(\mathsf{pk}_i, G_{new})$ and S_{new}.

  After receiving messages at the end of round $f+1$, process P_i performs steps 1–4 above (as though in round $f+2$) and outputs the set $V_i := accepted$.

Figure 1: Our ISC protocol for the $\mathcal{F}_{puz}$-hybrid model.

Intuition behind the protocol. Our protocol is closely related to the Dolev-Strong protocol for Byzantine agreement (assuming a pre-established PKI), and follows the same communication pattern: in each round r, a process *accepts* a value if it has received a collection of r signatures on that value; the process then adds its own signature to the collection and relays it to the other processes. However, since in our setting we do not have a PKI and processes do not know each other's public key, we must add an additional constraint (based on the $\mathcal{F}_{puz}$ functionality) that prevents the adversary from utilizing more than one keypair per corrupted process. Our constraint is based on the observation that the adversary can solve at most f puzzles in one round, and any depth-ℓ subgraph in a puzzle graph received in round r must have been solved in round $r-\ell$ (due to the time-lock nature of the puzzles modeled by $\mathcal{F}_{puz}$). Thus in our protocol, a correct process only considers a public key "valid" if it comes along with a puzzle graph containing that public key at sufficient depth.

Our protocol is defined in Figure 1. We now prove that it realizes ISC.

Lemma 1. *For every correct process P_i, its identity pk_i is accepted by every correct process in* *round* 2.

Proof. This follows immediately from the protocol definition. In the first round, each correct process P_i broadcasts its own depth-1 puzzle graph and signature on pk_i, and every other process receives these messages and accepts pk_i in the next round. $\square$

Lemma 2. *If a correct process P_i accepts a key pk in round $r < f + 1$, then every correct process accepts pk in round $r + 1$ or earlier.*

Proof. The proof is by induction on the round number $r \geq 1$. For the base case, when $r = 1$ observe that each correct process P_i only accepts its own key pk_i, and by Lemma 1 every other correct process accepts pk_i in round 2.

Suppose the lemma holds at round r, and suppose a correct process P_i accepts pk in round r.

First, we will prove that in round $r+1$, every other correct process receives at least r signatures over pk from distinct previously-accepted keys. P_i must have received signatures on pk from $r - 1$ keys (not including its own), all of which must have been accepted by P_i prior to round r. Therefore, applying the inductive hypothesis, every other correct process must have accepted these $r-1$ prior to round $r+1$ (this holds vacuously in the case that $r = 1$). P_i creates and publishes an additional signature on pk using its own key pk_i. By Lemma 6, every correct process has already accepted pk_i by the end of round 2. Therefore in round $r+1$, every correct process receives at least r signatures on pk from keys it has accepted.

Second, observe that if P_i accepts pk in round r, then it must have received a graph g containing a depth-$(r-1)$ subgraph h such that $\mathsf{pk}(h) = \mathsf{pk}$. By solving an additional puzzle, P_i obtains a puzzle graph $g_{r,i}$ in which h is at depth r. Since every correct party receives $g_{r,i}$ in round $r + 1$, every correct party accepts in that round. $\square$

Lemma 3. *Each correct process accepts at most n distinct keys.*

Proof. Fix some $r \geq 1$. If a correct process accepts a key pk in round $r + 1$, then it must have received a puzzle graph g containing a depth-r puzzle graph. In the time-lock puzzles model, each corrupted party can solve only one puzzle per round; therefore, the puzzle associated with any depth-r puzzle graph received in round $r + 1$ must have been computed during round 1. Since at most n puzzle solutions can be found in a single round, at most n distinct keys are ever accepted. $\square$

Theorem 1. *For any $f < n$ there is a polynomial-time ISC protocol with $f + 1$ rounds of communication, secure against f corrupted parties.*

Proof. The Boundedness property follows from Lemma 3. Validity follows directly from Lemma 6.

To prove the Agreement property, we must prove that if a correct process accepts a key pk in the final round $f + 1$, then every correct process has accepted pk by round $f + 1$ or earlier. Suppose correct process P_i accepts pk in round $f + 1$. Then process P_i must have received $f + 1$ signatures on pk from previously-accepted keys (not including its own). Observe that among these $f + 1$ previously-accepted keys, at least 1 must belong to some honest party P_j: by Lemma 1, the keys of all $n - f$ correct processes are accepted by P_i in round 1, and by Lemma 3 at most n keys in total are accepted among all the correct processes. Since a correct process only signs pk after accepting it, P_j must have accepted pk in round f or earlier. Therefore, by Lemma 2, if P_j accepts pk in round $f + 1$, then every correct process accepts pk in round $f + 2$ or earlier. $\square$

Message Complexity. In the worst case, each process publishes n^2 signed messages, n for each accepted key. Since no more than $n(f+1)$ puzzle solutions are solved in total, any puzzle graph can be represented using only $O(\lambda nf)$ bits (i.e., the graphs should be represented in a way that avoids duplicating shared children). Therefore since each process accepts at most n keys, it publishes at most $O(\lambda n^2 f)$ message bits. As a further optimization, each process may keep track of which subgraphs it has already published (e.g., after accepting a key in an earlier round) and avoid publishing duplicates. This reduces the worst case message cost to $O(\lambda nf)$ bits per process.

3.1 A Lower Bound on the Round Complexity

The ISC protocol described above uses an *optimal number of rounds* for *compliant* protocols (i.e., protocols that are essentially deterministic, except for their use of digital signatures and interactions with the puzzle oracle). Our lower bound proof is inspired by the Dolev-Strong lower bound for authenticated Byzantine agreement [12], but requires non-blackbox modifications. In particular, we show that the same lower bound still holds even with the additional bounded-rate $\mathcal{F}_{puz}$ oracle, despite the fact that this $\mathcal{F}_{puz}$ oracle can be used to further constrain the adversary's behavior. Our overall approach is based on a simple bivalency proof of the original theorem [1].

In Appendix A, we prove the following lower bound on the round complexity of any compliant ISC protocol:

Theorem 2 (Round complexity lower bound.)**.** *Suppose that the number of correct processes is at least* 2 *(since otherwise the problem is trivialized), and that* $n > f + 1$*. Any* compliant *ISC protocol in the* $\mathcal{F}_{puz}$*-hybrid model of computation that tolerates* f *faults must have at least* $f + 1$ *rounds.*

4 Pseudonymous MPC from Time-lock Puzzles

In the previous section, we proved that our protocol (Figure 1) solves the ISC problem in the $\mathcal{F}_{puz}$ network model. This allows the participating parties to agree on a set of public keys, and honest parties' public keys are guaranteed to be included; effectively, this bootstraps a pseudonymous PKI.

In this section, we show that the ISC protocol can be utilized to achieve (pseudonymous) secure computation for general tasks. In particular, we show that after running ISC, the pseudonymous PKI can subsequently be used to implement a pseudonymous version of secure function evaluation (SFE). Our approach is to first show that a standard algorithm (Dolev-Strong [12]) can be composed with ISC in order to implement a pseudonymous version of *secure broadcast* (also known as Byzantine agreement), and then finally composed with a standard broadcast-based algorithm (BGW [6]), leading to SFE.

4.1 Why Pseudonymous Security?

Standard definitions of secure multi-party computation implicitly assume that each party has a pre-established "real identity". Without loss of generality, assume that the parties' real identifiers are $\{1, 2, \ldots, n\}$, then the ideal functionality will collect everyone's private inputs $\{x_1, \ldots, x_n\}$, compute the function $(y_1, \ldots, y_n) := f(x_1, \ldots, x_n)$ and give y_i to the i-th party. Note that the corresponding indices for the inputs and outputs are bound to parties' pre-established identities.

This model, however, is inherently too strong for peer-to-peer networks like Bitcoin without PKI or pre-established trust between peers. We formally show this in Claim 1 below, which motivates the "pseudonymous" relaxation of standard security definitions.

Claim 1. *There is no protocol that simulates standard authenticated channels (i.e., secure evaluation of the function $f(x_1, x_2, ..., x_n) = (x_1, x_2, ..., x_n)$) in the $\mathcal{F}_{puz}$ model, if there is even one corruption.*

Proof. The attack is as follows: the adversary corrupts one process, and uses the corrupted process to simulate the correct behavior with a random input value. Let the random coins used by the honest party P_i be r, and let the random coins of the corrupt party be r'. The key observation is that nothing in the $\mathcal{F}_{puz}$ model (i.e., neither the communication channel nor the puzzles) allow any observable difference based on the party ids. Therefore the outcome when the random coins are flipped (i.e., P_i has random coins r' and the corrupted party has r) are identical. □

4.2 Definition of Pseudonymous Security

Pseudonymous security captures the fact that every party in such a peer-to-peer-like network can claim any pseudonym but must stick to the same pseudonym through out the computation. This relaxed notion of secure computation is useful in many circumstances. First, many functions of interest (such as majority value, average value, etc.) are invariant under any permutation of the input values, and therefore unaffected by this relaxation. Second, in many scenarios the identity of the other participants may simply be irrelevant. For example, in a game of online poker game against an anonymous opponent, one might only be concerned that his opponent has deposited the correct amount of valid digital currency.

To define pseudonymous SFE, *we allow the adversary to choose an arbitrary permutation on the inputs.* Parties must learn their own position in the permutation (as well as the corresponding output value), but may be unable to the permuted index of any of the other parties. We begin with the definition in [2, 8] for standalone security against a non-adaptive adversary, and modify it as necessary to match our model. The definition is based on an experiment involving a pair of execution models, one for the *ideal world* (in which a hypothetical trusted third party collects the parties' inputs and computes the function) and one for the *real world* (in which parties execute the secure protocol themselves). For the real world, we must modify the communication mechanism to reflect our message diffusion primitive, rather than built-in point-to-point channels; we must also add in the time-lock puzzle mechanism. For the ideal world, the key modification is that we allow the adversary to choose a permutation, and then the function is evaluated on the permuted input vector rather than the original vector, and parties receive the corresponding element of the permuted output vector. The details of this execution model are elaborated below.

In the ideal world, we need to modify the ideal process as described in [2, 8] to include a mechanism for the adversary to permute the inputs and outputs.

Let $f : \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$ be a function to evaluate. Each party P_i is given input $x_i \in \{0,1\}^n$ and a security parameter λ. The parties wish to evaluate $f(\lambda, \vec{x}, r)_1, ..., f(\lambda, \vec{x}, r)_n$, where $r \xleftarrow{\$} \{0,1\}^s$ is a random string and s is a function of the security parameter. The output for party P_i is the corresponding element of the output, $f(\lambda, \vec{x}, r)_i$, which is computed by a trusted party T. The ideal-world adversary $\mathcal{S}$ is an interactive Turing machine that controls the corrupted parties. We are modeling non-adaptive corruptions; the set of indices of corrupted parties is indicated by $C \subset \{1, ..., n\}$, and the set of uncorrupted parties is indicated by I. The adversary $\mathcal{S}$ receives the inputs and random coins drawn by the corrupted parties, as well as an auxiliary input string z and the security parameter λ. The ideal process proceeds as summarized in Figure 2.

Ideal-world Execution for evaluating function f with adversary $\mathcal{S}$

- **Input substitution:** The ideal-process adversary $\mathcal{S}$ sees the inputs of the corrupted parties. $\mathcal{S}$ is active and may alter these inputs based on the information known to it so far. Let $\vec{b}$ be the $|C|$-vector of the altered inputs of the corrupted parties, and let $\vec{y}$ be the n-vector constructed from the input $\vec{x}$ by substituting the entries of the corrupted parties by the corresponding entries in $\vec{b}$.
- **Permutation:** The ideal-process adversary $\mathcal{S}$ chooses a permutation $perm \in Perm([n])$ of the party indices.
- **Computation:** Each party P_i passes its (possibly modified) input value, y_i, to the trusted party T. Let $\vec{y}'$ be the vector obtained by applying the permutation $perm$ to the inputs (i.e., $\vec{y}'_{perm(j)} = \vec{y}_j$, for each $j \in [n]$). Next, T chooses $r \stackrel{\$}{\leftarrow} \{0,1\}^s$, and passes each P_i the value $perm(i)$, (i.e., its position in the permutation) and the value $f(\lambda, \vec{y}', r)_{perm(i)}$.
- **Output:** Each uncorrupted P_i outputs $perm(i)$ and $f(\lambda, \vec{y}', r)_{perm(i)}$, and the corrupted parties output $\perp$. In addition, the adversary outputs some arbitrary function of the information gathered during the computation in the ideal process. This information consists of the adversary's random input, the corrupted parties' inputs, the entire permutation $perm$, and the resulting function values $\{f(\lambda, \vec{y}', r)_{perm(i)} : P_i \text{ is corrupted}\}$.

Figure 2: Summary of ideal-world execution model for secure function evaluation

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

1. Receive from each party P_i an input $(\texttt{solve}, x^{(i)})$. (Note that only a *single* value is allowed.) For $i = 1, \ldots, n$, first check if a pair $(x^{(i)}, h^{(i)})$ has been stored in T, and if so return $h^{(i)}$ to P_i; otherwise, choose uniform $h^{(i)} \in \{0,1\}^\lambda$, return $h^{(i)}$ to P_i, and store $(x^{(i)}, h^{(i)})$ in T.
2. Receive from each party P_i an arbitrary-length vector $(\texttt{check}, (x_1^{(i)}, h_1^{(i)}), \ldots)$. Return to each party P_i the vector of values $(b_1^{(i)}, \ldots)$ where $b_j^{(i)} = 1$ iff $(x_j^{(i)}, h_j^{(i)}) \in T$.

Real-world ($\mathcal{F}_{puz}$-hybrid) Execution with adversary $\mathcal{A}$

1. (a) Each party P_i starts with the security parameter λ, input x_i, and random input r_i.
   (b) The adversary $\mathcal{A}$ starts with λ, random input r_0, input z that includes a set $C \subset [n]$ of corrupted parties and their inputs $\{x_i | i \in C\}$, and additional auxiliary input.
2. Initialize a round counter, $l := 0$.
3. As long as there exists an uncorrupted party that did not halt, repeat:
   (a) Each uncorrupted party $P_i, i \notin C$, performs a compute phase during which it may interact with an instance of $\mathcal{F}_{puz}$ using its oracle tape, subject to the constraint that it may make at most one oracle query of the form $(\texttt{solve}, x)$ in this round. Note that the responses to these queries are not delivered until every query for this round has been made.
   (b) Each uncorrupted party $P_i, i \notin C$, generates $\{m_{i,l}\}$, where each $m_{i,l} \in \{0,1\}^*$ is a (possibly empty) message intended to be published during this round.
   (c) The adversary $\mathcal{A}$ learns $\{m_{i,l} | i \in [n]\}$, and generates $\{m^\dagger_{j,l} | j \notin C\}$, a set of messages to be delivered just to P_j. During this phase, the adversary may make up to $|C|$ oracle queries of the form $(\texttt{solve}, x)$.
   (d) Each uncorrupted party P_i, $i \notin C$, receives the messages $sort(m^\dagger_{i,l} \cup \{m_{j,l} | j \notin C\})$.
   (e) $l := l + 1$.
4. Each uncorrupted party $P_i, i \notin C$, as well as $\mathcal{A}$, generates an output. The output of the corrupted parties is set to $\perp$.

Figure 3: A summary of the nonadaptive $\mathcal{F}_{puz}$-hybrid computation.

We denote by $\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x})$ the random variable corresponding to the output of each party P_i, as well as the output of the ideal-world adversary $\mathcal{S}$.

The real-world execution is summarized in Figure 3. We use $\text{REAL}_{f,\mathcal{A}(z),\pi,I}(\vec{x})$ to denote the random variable consisting of the output values of each protocol party P_i, as well as the view of the adversary $\mathcal{A}$.

Definition 5. ***(Pseudonymous Multi-party Computation.)*** *Let $\mathcal{P} = \{P_i\}_{i=1}^n$ denote a set of parties, let $f : \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$ be a function, and let $\vec{x} \in (\{0,1\}^*)^n$ denote a vector of input values associated with each party. A protocol π* pseudonymously computes *f if for every real-world adversary $\mathcal{A}$, there exists an ideal-world simulator $\mathcal{S}$, such that:*

$$\text{IDEAL}_{f,\mathcal{S}(z),I}(\vec{x}) \approx \text{REAL}_{f,\mathcal{A}(z),\pi,I}(\vec{x}).$$

4.3 Protocol for Pseudonymously Secure Multi-party Computation

The BGW protocol for multi-party computing makes use of two communication primitives: private authenticated channels, and broadcast. We will need to emulate (the pseudonymous version of) these in the $\mathcal{F}_{puz}$ model.

For private channels, the parties must generate a public-key encryption keypair (in addition to the signing key) at the outset, and then perform the ISC protocol using their public key as their input value. Next, private messages can be sent to each pseudonym-holder by using standard techniques to sign and encrypt the message, and publishing the ciphertext using the underlying message diffusion primitive.

Since, at the end of ISC, each party has a signing key and a list of public keys serving as pseudonyms for the other parties, it is possible to run existing protocols that make use of a standard PKI, such as the Dolev-Strong algorithm for Byzantine agreement (aka, secure broadcast). Arbitrary subsequent instances of broadcast can be executed, making use of the synchronous message diffusion channel and the keypairs established in the initial execution of ISC, but without any further need to use the time-lock puzzles. We note that in order to compose executions of Byzantine agreement, it is necessary to use a unique *session identifier* for each execution; this prevents signatures on messages in one execution from being *replayed* in another execution. [27]

The pseudonymous protocol for securely evaluating a function f proceeds as follows:

1. First, each party P_i generates a signing and encryption keypair, pk_i and sk_i. Next, each party executes the ISC protocol 1, using pk_i as its input value. At the end of the protocol, each party obtains a vector $\vec{\mathsf{pk}}$ of *permuted* public keys (in sorted order), in which pk_i is an element. Let j be the index such that $\vec{\mathsf{pk}}_j = \mathsf{pk}_i$ (i.e., j is the pseudonym of P_i).

2. Next, each party P_i executes the BGW protocol as in [6] though with index j. Note that the BGW protocol uses both broadcasts and private messages. When the protocol calls for a broadcast, the Dolev-Strong algorithm is executed using $\vec{\mathsf{pk}}$ as the set of signing public keys, as described above. When the protocol calls for a private message, the signing and encryption keys are used as described above. Upon completion of the BGW protocol, party P_i outputs its pseudonym j as well as the function output value it receives.

Theorem 3. (*Pseudonymously Secure SFE.) In the $\mathcal{F}_{puz}$-hybrid model, we can achieve SFE for general polynomial functions while ensuring pseudonymous security against a non-adaptive adversary. Specifically, input completeness and guaranteed termination can be achieved with honest* *majority, i.e., $f < n/2$. In the presence of arbitrary number of corruptions, we can ensure security with unanimous abort.*

Proof. First we prove that the protocol described above satisfies the case for honest majority. Since the underlying BGW protocol is executed intact, the existing simulator for this protocol, $\mathcal{S}'$, can be reused, with the permuted player indices substituted for the correct indices. The main thing for us to show is that the real-world execution of the ISC protocol in the first phase can indeed be simulated.

Our simulator $\mathcal{S}$ first generates keypairs pk_i and sk_i for each simulated party. The view of the real-world adversary consists of the messages sent by the honest parties in each round; this communication pattern can be simulated perfectly. According to the *agreement* property of the ISC definition, if a corrupted party does not complete the protocol, its public key may be replaced with $\perp$; in any case, all uncorrupted parties arrive at a consistent vector $\vec{\mathsf{pk}}$ with high probability. Since the ISC protocol sorts this vector in lexicographical order, the simulator $\mathcal{S}$ instructs the trusted party T to use this permutation.

After completing the ISC phase, our simulator $\mathcal{S}$ executes the BGW simulator $\mathcal{S}'$ in a sandbox, placing each party P_i with the corresponding permuted index. Note that since $\mathcal{S}'$ is intended for a model with broadcast and private channels, we must be able to simulate the protocols used to implement these channels. It is trivial to simulate the broadcast protocol since the simulator already knows the signing keys; the private channel can be trivially simulated by encrypting random strings.

The BGW protocol, however, provides security only when an honest majority of participants is assumed; for example, the adversary can learn the private inputs of the parties when a majority of participants are corrupted. It is possible to compose our ISC protocol with other alternatives as well, such as GMW [19], which guarantees privacy regardless of the number of corruptions. We omit the details of this; the protocol and proof for composing ISC with GMW proceed exactly as with BGW. □

Adaptive Security Finally, we note that although for simplicity we have presented our results for a non-adaptive adversary, our technique would also be applicable for an adaptive adversary. A subtle issue arises when implementing secure broadcast in the adaptive setting, depending on a) whether broadcast is defined to allow the adversary to corrupt the broadcast sender *after* learning the value it attempts to send, and b) whether the underlying communication channel can be interrupted [22]. If our diffusion channel is atomic and it is necessary to prevent an interrupted broadcast against a dishonest majority, then the technique of [17] could be used.

5 Conclusions and Future Work

This work is inspired by peer-to-peer networks like Bitcoin, and makes an initial attempt to understand what meaningful security properties can be achieved in such distributed networks. Contrary to the wide-held belief that "nothing interesting" can be achieved in such networks without authenticated channels or PKI, we show that by placing a strict bound on each party's computational resource, we can achieve a *pseudonymous* notion of secure computation for general functions. Although our work does not directly reason about the security of the de facto Bitcoin protocol, our result shows that distributed computation models resembling Bitcoin can lead to rich applications with non-trivial security guarantees, where security is bootstrapped from the bound on each party's computation resource.

This paper opens up numerous avenues for future research. In our work, we have made several simplifying assumptions, and relaxing any of them is an important next step. For example, we have assumed that there is a fixed number n of parties that is public knowledge at the outset of the protocol. Is anything achievable if n is unknown? Alternately, what can be said in a *partially synchronous* network where the maximum communications delay is unknown? In a different direction, it would be interesting to explore a model in which parties were not *malicious* or *honest*, but are instead only *rational*. Here, we could assign some "cost" to solving puzzles rather than assuming a strict upper bound on how many puzzles a party could solve per round. This is the direction taken in other work analyzing Bitcoin [14, 24]. It would be useful to reconcile this with approaches to modeling rationality in secure multi-party computation.

References

[1] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires t+ 1 rounds. *Information Processing Letters*, 71(3):155–158, 1999.

[2] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. In *Electronic Colloq. on Comp. Complexity (ECCC)*, 2011.

[3] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical report, Computer Science Dept., Yale University, 2005.

[4] Adam Back. Hashcash—a denial of service counter-measure. `http://www.hashcash.org/papers/hashcash.pdf`, 2002.

[5] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *Advances in Cryptology—CRYPTO 2005*, pages 361–377. Springer, 2005.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM Symposium on Theory of computing*, pages 1–10. ACM, 1988.

[7] Nikita Borisov. Computational puzzles as sybil defenses. In *Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 171–176. IEEE, 2006.

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

[9] Fabien Coelho. An (almost) constant-effort solution-verification proof-of-work protocol based on merkle trees. In *Progress in Cryptology–AFRICACRYPT 2008*, pages 80–93. Springer, 2008.

[10] Jeffrey Considine, Matthias Fitzi, Matthew Franklin, Leonid A Levin, Ueli Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *Journal of Cryptology*, 18(3):191–217, 2005.

[11] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Anne-Marie Kermarrec, Eric Ruppert, et al. Byzantine agreement with homonyms. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 21–30. ACM, 2011.

[12] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[13] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer Verlag, 1993.

[14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, 2014.

[15] Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional Byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 482–501. Springer Verlag, 2002.

[16] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable Byzantine agreement secure against faulty majorities. In *21st ACM Symposium Annual on Principles of Distributed Computing*, pages 118–126. ACM Press, 2002.

[17] Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. pages 179–186. ACM, 2011.

[18] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Technical report, 2014.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM Press, 1987.

[20] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[21] Bogdan Groza and Bogdan Warinschi. Cryptographic puzzles and DoS resilience, revisited. *Designs, Codes and Cryptography*, 2013.

[22] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology–EUROCRYPT 2010*, pages 466–485. Springer, 2010.

[23] Ari Juels and John G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, volume 99, pages 151–165, 1999.

[24] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining or, bitcoin in the presence of adversaries. In *Workshop on Economics in Information Security (WEIS)*, 2013.

[25] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[26] Ben Laurie and Richard Clayton. Proof-of-work proves not to work. In *Workshop on Economics and Information Security*, 2004.

[27] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated Byzantine agreement. *J. ACM*, 53(6):881–917, 2006.

[28] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 373–388. ACM, 2013.

[29] A Miller, A Juels, E Shi, B Parno, and J Katz. Permacoin: Repurposing bitcoin work for long-term data preservation. *IEEE Security and Privacy*, 2014.

[30] Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin, 2014.

[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf`, 2008.

[32] Michael Okun. *Distributed Computing Among Unacquainted Processors in the Presence of Byzantine Failures.* PhD thesis, Hebrew University of Jerusalem, 2005.

[33] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[34] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.

A Lower Bound on Round Complexity in the $\mathcal{F}_{puz}$ Model

Dolev and Strong first showed that assuming authentication, any deterministic Byzantine agreement protocol tolerating f faults (where $n > f + 1$) must have at least $f + 1$ messaging rounds [12].

Our lower bound proof is inspired a simple bivalency-based proof of this theorem [1]. We basically must show that having the additional puzzle functionality does not change this lower bound, even though this additional functionality can be used to further restrict behavior of the adversary, since the adversary can only solve a limited number of computational puzzles per round.

For the purpose of our lower bound, we limit our concern only to *compliant* protocols (as defined below) that are effectively deterministic. We must carefully explain what this means in our setting, since the $\mathcal{F}_{puz}$ functionality itself is randomized and our positive results make use of digital signature algorithms.

Definition 6. *A* compliant *protocol in the $\mathcal{F}_{puz}$-hybrid model must ignore the security parameter and treat the public/private keys, digital signatures, and puzzle solutions as opaque strings. Protocols may transmit opaque strings along communication channels and include them as input to oracle queries and digital signature algorithms, but cannot otherwise inspect or modify their bits.*

This definition in particular rules out protocols that use the signature algorithm or interaction with $\mathcal{F}_{puz}$ in order to perform coin flips.

Theorem 2 (Round complexity lower bound.). *Suppose that the number of correct processes is at least* 2 *(since otherwise the problem is trivialized), and that* $n > f + 1$*. Any* compliant *ISC protocol in the* $\mathcal{F}_{puz}$*-hybrid model of computation that tolerates* f *faults must have at least* $f + 1$ *rounds.*

For the purpose of proving our lower bound, we limit ourselves to constructing adversaries that are also compliant in the same sense, and in fact only need to *crash* processes (rather than induce Byzantine behavior). Furthermore, our adversaries crash at most one process per round.

We will state a lemma that given these constraints, the same reasoning used in deterministic models applies equally to compliant protocols in our model with high probability. [2] First we define some notation useful for describing executions in our model.

A *partial run* is a finite sequence of system configurations (including the input values to each process and the internal state of each process and the adversary) that transition according to our model. The last element in an *r-round partial run* is the system configuration at the end of r rounds. Note that since our execution model is randomized (due to coin flips taken by the puzzle functionality and digital signature scheme), given an initial configuration our model defines a probability distribution over r-round partial runs.

We define an equivalence relation among partial runs in our model. Two partial runs are equivalent if they are equal up to a bijective transformation of keys, signatures, and puzzle solutions. A partial run s can be represented as a pair $(state, \{o_i\})$, where $\{o_i,\}$ is the sequence of every occurrence of an opaque string in a configuration and *state* is the sequence of system configurations with each occurrence of an opaque string taken out (and replaced with a symbol `opaque`). Another partial run s' is equivalent to s if $s' = (state, \{F(o_i)\})$ for some bijective function $F : \{0,1\}^* \to \{0,1\}^*$. This implies that two partial runs are equivalent if their initial conditions are the same, any decisions made by correct processes are the same, any processes that have crashed did so at the same time, and the content of the internal states of each process and messages delivered differ only in the representation of the opaque strings associated with signatures and puzzles.

Lemma 4. *Consider a compliant protocol and compliant adversary (as per Definition 6). Then for each round* r*, there exists a single equivalence class of* r*-round partial runs (which we call the* canonical *partial run) such that with high probability, a partial run sampled from the distribution of* r*-round partial runs belongs to this equivalence class.*

Proof. As the protocols make only a (polynomially) bounded number of queries (to $\mathcal{F}_{puz}$ or to digital signature routines), with high probability the processes do not generate colliding keys or puzzle solution strings, nor does any process "guess" a puzzle solution it has not received from $\mathcal{F}_{puz}$ or the communication channel. □

Hereafter, we are only concerned with the canonical partial runs (rather than partial runs that involve some unlikely string collision), and therefore the term *partial run* should be taken to mean the class of *canonical partial runs*.

Let s_r be an r-round partial run. We say that s_{r+1} is a *one-round extension* of s_r if there exists some adversary $\mathcal{A}$ such that s_r and s_{r+1} are respectively the canonical r-round and $r+1$-round

[2] Note that an alternative to invoking the above lemma would be to redefine digital signatures and our puzzle functionality to inherently use opaque "tags" rather than concrete bit-strings - indeed this is the approach taken in [12]. We have chosen our approach because it allows us to use a simple and concrete representation of the model for presenting our positive results.

partial runs given $\mathcal{A}$. A partial run s is an *extension* of s_r if it is a partial run obtained by extending s_r one round at a time and in which all processes have decided.

Having established a basis for applying deterministic reasoning to our model, the lower bound proof can be carried out with simple modifications to the bivalency proof in [1]. We describe these necessary modifications and then refer the reader to the text of [1] for the remainder of the proof.

The lower bound of [12, 1] applies to binary consensus, a somewhat different problem than ISC in which processes must decide on a single bit). We can imagine that each process decides on 2^λ bits, one for each possible input value (at most n of these may be set to 1). The proof in [1] relies on a notion of *valences*, which we now define in our setting. A partial run s_r is *v-valent* (for some value v) if in all extensions of s_r, the correct processes decide on a set S where $v \in S$, and *$\overline{v}$-valent* if in all extensions $v \notin S$. When v is clear from context, we abbreviate v-valent with 1-valent and $\overline{v}$-valent in order to more closely match the notation in [1]. A partial run is univalent (with respect to v) if it is either v-valent or $\overline{v}$-valent, and bivalent if it is neither v-valent nor $\overline{v}$-valent.

The proof in [1] involves constructing pairs of executions that are indistinguishable to some of the processes in the network. We next prove a lemma that guarantees that these constructions remain indistinguishable, even in our setting where the processes may additionally interact with $\mathcal{F}_{puz}$.

Lemma 5. *For $r < f$, let s_r be an r-round partial run in which at most r processes have crashed. Consider the following two one-round extensions of s_r: s_{r+1}, in which one process crashes* before *delivering its message to some process p, and s'_{r+1} in which the same process crashes* after *delivering its message to p. Let s' and s denote the runs extending s'_{r+1} and s_{r+1} respectively in which p crashes at the beginning of the next round (before sending any messages). (Note that in the special case $r+1 = f$, the processes terminate so the crash is unnecessary.) Let p' (distinct from p) be a correct process. Process p' cannot distinguish between runs s and s' (i.e., its behavior in both runs is the same).*

Proof. In the case of an ordinary message passing network, this is immediate, since p is the only process that observes any difference in round $r + 1$ and p crashes before it can communicate to any other process. However in our setting we must prove this holds in spite of the additional $\mathcal{F}_{puz}$ functionality, since in round $r + 1$, process p may affect $\mathcal{F}_{puz}$ by solving a puzzle before crashing. Without loss of generality, assume that in run s, p solves a puzzle puz and in s' solves a puzzle puz'. However, note that a) $\mathcal{F}_{puz}$ records a puzzle solution sol randomly sampled from a space of size 2^λ, b) the recorded solution is returned only to process p, which crashes before it can transmit any information about it to another process, and c) the other processes may only observe the difference through interaction with $\mathcal{F}_{puz}$ by calling sending $(\mathtt{check}, \mathsf{puz}, \mathsf{sol})$ or $(\mathtt{check}, \mathsf{puz}', \mathsf{sol})$ (i.e., by correctly *guessing* the puzzle solution). As the processes only make a polynomial number of queries, this occurs with negligible probability. □

Now that we have defined the notions of (canonical) partial runs and valences in our setting and established conditions under which indistinguishability holds despite the additional $\mathcal{F}_{puz}$, our theorem follows from the proof in [1] verbatim.

B ISC in the Proof-of-Parallelizable-Work Model

While in our main result we have considered inherently sequentially puzzles (i.e., puzzles that take the adversary an entire round to solve, even with all f corrupted processes working together in parallel), we are also interested in modeling puzzles that the adversary can solve faster using its parallel resources. Equivalently, we may be interested in modeling an adversary that can compute sequentially f times faster than a corrupt process.

We can model this via a modified functionality $\mathcal{F}_{parpuz}$, which differs from $\mathcal{F}_{puz}$ in that it allows the adversary to make multiple rounds of interaction with the functionality within a single communication round.

- Receive from each *uncorrupted* party P_i an input $(\texttt{solve}, x^{(i)})$. (Note that only a *single* value is allowed.) For $i = 1, \ldots, n$, first check if a pair $(x^{(i)}, h^{(i)})$ has been stored in T, and if so return $h^{(i)}$ to P_i; otherwise, choose uniform $h^{(i)} \in \{0,1\}^\lambda$, return $h^{(i)}$ to P_i, and store $(x^{(i)}, h^{(i)})$ in T.

- For up to f iterations, a single corrupted party P_i may request $(\texttt{solve}, x_i)$, and the input is processed immediately as above.

- Receive from each party P_i an arbitrary-length vector $(\texttt{check}, (x_1^{(i)}, h_1^{(i)}), \ldots)$. Return to each party P_i the vector of values $(b_1^{(i)}, \ldots)$ where $b_j^{(i)} = 1$ iff $(x_j^{(i)}, h_j^{(i)}) \in T$.

As before, we allow parties to call $\mathcal{F}_{parpuz}$ with a `check` instruction any (polynomial) number of times. Each of the honest parties is allowed to call $\mathcal{F}_{parpuz}$ with a `solve` instruction only once per communication round; moreover, all of the `solve` instructions for a round must be sent before any honest party receives its puzzle solution. However, the corrupted parties can call $\mathcal{F}_{parpuz}$, one after another in sequence, up to a total of f times within an overall communication round.

Protocol Intuition for the $\mathcal{F}_{parpuz}$-Hybrid Model. Our protocol proceeds in two overall phases. In the first phase, called the "mining" phase, each correct process constructs a chain of $O(f^2)$ puzzle solutions associated with that process's public key. In the second phase, the "communication" phase, the processes publish their puzzle chains and propagate the puzzle chains they receive from others. To ensure agreement, each process also signs and relays signatures according to the Dolev-Strong algorithm. Signatures corresponding to public keys without associated puzzle chains are ignored. The communication phase ends after $f+1$ rounds.

Intuitively, the protocol works because every correct process is able to create a valid puzzle chain for its own key, yet the corrupt processes are only able to create at most f puzzle chains before the protocol terminates.

Definition 7. *A* length-ℓ puzzle chain *is defined inductively as follows:*

- *A puzzle graph g is a length-1 puzzle chain if* $\mathsf{children}(g) = \emptyset$.

- *A puzzle graph g is a length-$(\ell+1)$ puzzle chain if* $\mathsf{children}(g) = \{s\}$, $\mathsf{pk}(s) = \mathsf{pk}(g)$, *and s is a length-ℓ puzzle chain.*

The protocol is defined in Figure 4.

An $O(f^2)$-Round ISC Protocol using Parallelizable Puzzles

Initially, each process P_i generates a signing keypair $(\mathsf{sk}_i, \mathsf{pk}_i)$ according to some digital signature scheme, where pk_i is the concatenation of the underlying public key and the i^{th} input value. Hereafter we refer to pk_i as the identity of process P_i. The subroutines *solve* and $sign_i$ are the same as in 1.

- **Mining Phase.** In the first round, each process P_i constructs a puzzle graph

$$g_{i,1} := solve(\mathsf{pk}_i, \emptyset)$$

where pk_i denotes the sender's identity, and the message includes an empty history. In each of the following rounds, up to round $r_{mine} = f(f+1)+1$, each process extends its chain by one puzzle solution

$$g_{i,r} := solve(\mathsf{pk}_i, \{g_{i,r-1}\}).$$

The mining phase ends after r_{mine} rounds. We abbreviate $g_{i,r_{mine}}$ by g_i.

- **Communication Phase.** Initially, in round 1 of the communication phase, each process P_i

  1. sets $accepted := \{\mathsf{pk}_i\}$,
  2. publishes g_i,
  3. and publishes $sign_i(\mathsf{pk}_i)$.

  Thereafter, in each round $2 \leq r \leq f+2$, process P_i

  1. Receives a set of signed messages S, such that $\forall s \in S$, $\mathsf{pk}(s) \in accepted$ and s has a distinct pk and m (i.e., $\forall s' \in S \backslash \{s\}, (\mathsf{pk}(s), msg(s)) \neq (\mathsf{pk}(s'), msg(s'))$). Invalid or redundant signatures are discarded.
  2. Receives a set of length-r_{mine} graphs G such that $\forall g \in G$, $\mathsf{pk}(g) \notin accepted$, and each g has a distinct id (i.e., $\forall g' \in G \backslash \{g\}, \mathsf{pk}(g) \neq \mathsf{pk}(g')$). Redundant or invalid puzzle chains are discarded.
  3. For each $g \in G$, let $S_g := \{s \in S | id(s) = id(g)\}$. If $|S_g| \geq r$, then P_i:
     (a) sets $accepted := accepted \cup \{\mathsf{pk}(g)\}$,
     (b) publishes g,
     (c) and publishes $S_g \cup \{sign_i(\mathsf{pk}(g))\}$.

  At the end of round $f+2$ (of the communication phase), process P_i outputs the set *accepted*.

Figure 4: Our ISC protocol for the $\mathcal{F}_{parpuz}$-hybrid model.

Lemma 6. *For every correct process P_i, its identity pk_i is accepted by every other correct process in round 2.*

Proof. This follows immediately from the protocol definition. In round 1 of the communication phase, each correct process P_i broadcasts its own puzzle chain and signature on pk_i, and every other process receives these messages and accepts pk_i in the next round. □

Lemma 7. *If a correct process P_i accepts a key pk in round $r < f+2$ (of the communication phase), then every correct process accepts pk in round $r+1$ or earlier.*

Proof. The proof is by induction on the round number $r \geq 1$. For the base case, when $r = 1$ observe that each correct process P_i only accepts its own key pk_i, and by Lemma 6 every other correct process accepts pk_i in round 2.

Suppose the lemma holds at round r, and suppose a correct process P_i accepts pk in round r.

First, note that in round $r+1$, every other correct process receives at least r signatures over pk from distinct previously-accepted keys (see the proof of Lemma 2).

Second, observe that P_i publishes a length-r_{mine} chain g for pk in round r, and therefore every other correct party receives g in round $r+1$. Thus, every correct process accepts pk in round $r+1$, completing the proof by induction. □

Lemma 8. *Among the correct processes, at most n distinct keys are ever accepted.*

Proof. Each accepted key requires a length-r_{mine} chain, and each node in the chain must be associated with a consistent key. Each uncorrupted process only solves puzzles associated with its own key. Since the corrupted processes can solve (in total) at most f puzzles per round, they can solve at most $f(r_{mine} + f + 1)$ puzzles before the protocol terminates. Therefore, at most $\lfloor f(r_{mine}+f+1)/r_{mine} \rfloor = f + \lfloor f \cdot (f+1)/r_{mine} \rfloor = f$ length-r_{mine} chains can be computed by corrupt processes. Therefore, length-r_{mine} chains are found corresponding to at most n distinct keys. □

Theorem 3. *There exists a polynomial-time ISC protocol with $O(f^2)$ rounds of communication, secure against $f < n$ number of corrupted parties.*

Proof. The proof of this theorem is identical to that of Theorem 3, substituting Lemmas 6,7, and 8 for Lemmas 1,7, and 3. □