Pseudonymous Broadcast and Secure Computation from Cryptographic Puzzles

JONATHAN KATZ [*†] ANDREW MILLER[*] ELAINE SHI[‡§]

Abstract

In standard models of distributed 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 assumption that each party can solve cryptographic puzzles only at a bounded rate (and the existence of digital signatures), 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 parties are identified by pseudoynms rather than by their true identities.

Keywords: computational puzzles, distributed systems, consensus, secure multi-party computation

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

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

[‡]Cornell University. `runting@gmail.com`

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

1 Introduction

Standard models of distributed computing assume authenticated point-to-point channels between parties, where authentication may be provided via some physical property of the underlying network or using keys shared by the parties in advance. When security against a large fraction of corruptions is desired, even stronger pre-existing setup—e.g., a broadcast channel or a public-key infrastructure (PKI) with which broadcast can be implemented—is often 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 (see, e.g., [5, 32]), and in fact there are known impossibility results to this effect. For example, in the setting of multi-party computation, Barak et al. [5] show limits on what can be achieved if authenticated channels are not available. Even when authenticated channels are available, there are well-known bounds [25, 33] on the fraction of corrupted parties that can be tolerated without some form of setup such as a PKI [13].

The recent phenomenal success of Bitcoin [31] and other decentralized cryptocurrencies, however, has stirred vibrant interest in a new distributed-computing model where parties do not have known identities or pre-existing setup. Bitcoin offers evidence that interesting security properties *can* be achieved under such conditions. In this paper, we carry out a theoretical study of Bitcoin-like networks and show that, indeed, a wide class of tasks can be realized without setup, but based instead on *cryptographic puzzles* and precise bounds on an attacker's computational resources.

1.1 Results and Contributions

We make the following contributions.

Interactive set consistency and pseudonymous PKI. We consider a network that provides guaranteed delivery, but no authentication. We also assume the existence of cryptographic puzzles that can be solved at some bounded rate (but for which verification is easy), plus secure digital signatures. We show that these assumptions can be leveraged to allow a set of n parties to establish a "pseudonymous PKI" in the presence of *an arbitrary number of corrupted parties.* This means that all honest parties agree on a set of public keys (*agreement*) that contains each honest party's public key (*validity*) and at most n public keys overall (*boundedness*). We obtain a pseudonymous PKI rather than a true PKI because there is no binding between public keys and (external) identities.

In designing a protocol achieving the above, we face several challenges. Because there is no authentication, and no bound on the number of messages the attacker can send, the adversary can try to introduce fake identities ("Sybils") to overwhelm the honest parties. Intuitively, we defeat this by using computational puzzles to throttle the number of identities that can be claimed. Since the attacker can send different messages to different honest parties, the attacker can try to cause honest parties to reach inconsistent decisions. We defeat this sort of attack using techniques reminiscent of the Dolev-Strong protocol for authenticated Byzantine agreement [13]. We combine these ideas in a nontrivial way to construct our final protocol.

With a pseudonymous PKI in place, we can use protocols that rely on a (standard) PKI to achieve pseudonymous versions of various goals. For example, we can obtain pseudonymous authenticated communication by having parties sign the messages they send.[1] More interestingly, we can use the Dolev-Strong protocol [13] to realize pseudonymous broadcast. Coupled with our protocol for establishing a pseudonymous PKI in the first place, this shows that pseudonymous broadcast can be achieved without authenticated channels or prior setup in the presence of an unbounded number of corruptions, in contrast to known impossibility results for (standard) broadcast [25, 33].

Pseudonymous secure computation. Finally, we consider the more general task of secure multiparty computation. In a Bitcoin-like model where parties do not have well-known identities a priori, traditional notions of secure computation are not attainable. We are the first ones to formally define *pseudonymous* notion of secure computation in which, roughly speaking, inputs and outputs are bound to pseudonyms rather than parties' true identities. Pseudonymous secure computation can be realized in our model by using existing protocols (e.g., [20, 6]) and substituting pseudonymous authentication and broadcast for their standard counterparts.

Scope of our work. We stress that our work is *not* intended to model Bitcoin itself, but rather, to attain a basic theoretical feasibility understanding

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

about Bitcoin-like distributed systems. Such systems (whose security properties rely on computational resource allocation) have been to some extent overlooked by the traditional distributed systems and cryptography literature.

We readily admit that in our initial attempt to provide a clean, formal treatment we have made several simplifying assumptions. For example, we assume that the network guarantees (bounded-delay) delivery of messages to every party and that the number of parties is fixed. Based on these simplifying assumptions, we are able to rigorously formalize intuitions attained from Bitcoin-like cryptocurrencies, i.e., distributed-computation models based on assumptions about computational resources rather than authenticated identities can lead to non-trivial security guarantees. Our work poses numerous exciting directions for future research, including how to relax our assumptions to be more realistic, and how to design more efficient protocols.

1.2 Related Work

Cryptographic puzzles (also called *proofs of work*) have been analyzed and proposed for a variety of other purposes, including timed-release cryptography [34, 7], spam prevention [4, 15, 26], DoS resistance [22, 23], and defense against Sybil attacks [8]. Many proof-of-work puzzles (e.g., [10, 27, 4]) resemble variations of Merkle's hash trees, which are typically used for message authentication [28, 14]. In both applications, a pseudorandom function is used to determine a subset of branches that must be revealed; thus these constructions can often be used simultaneously as a puzzle and a signature.

Aspnes et al. [3] studied Byzantine agreement without a PKI in a model where computational puzzles can be solved at some bounded rate. That work assumes authenticated channels between honest parties; moreover, their feasibility results do not extend to an unbounded number of corruptions. Miller and LaViola, Jr. [30] and Garay et al. [19] present formal models in which the core Bitcoin protocol can be shown to achieve relaxed notions of consensus based on proof-of-work puzzles, but again under the assumption that a majority of the network is honest.

Okun [32] studied distributed-computing problems in several models of anonymous networks. The weakest model considered there, the "port-unaware" model, is most similar to ours. However, our model is weaker still: in the port-unaware model, each corrupt process can send at most one message to a correct process in each round, whereas in our model the adversary can deliver an arbitrary number messages to honest parties. Okun's positive results crucially rely on this message bound, and are thus inapplicable in our model. Similarly our communication model is related to (but weaker than) the models considered in [11, 12].

Our positive result 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 sets 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. We provide stronger security guarantees, albeit under a stronger assumption regarding the existence of cryptographic puzzles. 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 even with an unbounded number of corruptions.

Broadcast and secure computation without a PKI, but when 1/3 or more of the parties can be corrupted, is also studied in [17, 18, 21]. Some of these works 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.

Concurrent and independent work. Concurrently with our work, Andrychowicz and Dziembowski [2] show related results in a model that is similar to ours; however, we identify several differences. At a technical level, our models are incomparable. We assume the adversary cannot pre-compute puzzle solutions (in practice, this would be achieved using a random beacon to ensure that puzzles are now known until the protocol starts); they do not make this assumption, but instead assume a bound on the number of messages an adversary can send in each round. They consider puzzles for which finding solutions can be parallelized (called *parallelizable puzzles* in our work); we consider parallelizable puzzles as well as *sequential puzzles* for which parallel speedup is impossible. The round complexity of their protocol for establishing a pseudonymous PKI is $O(\kappa^2 \cdot f)$, where κ is a security parameter and f a bound on the number of corruptions, which is incomparable to the round complexity of our protocol based on parallelizable puzzles. Finally, they do not consider the notion of pseudonymous secure computation.

Garay et al. [19] analyze the security of the Bitcoin backbone protocol, and prove useful properties which they refer to as "common prefix" and "chain quality". Their model and result are incomparable to ours; in particular, the Bitcoin protocol they study utilizes only parallelizable puzzles, whereas we construct protocols using either parallelizable or sequentially hard puzzles, and in either case obtain better fault tolerance. They also do not make an effort to formalize secure computation in this model (which requires defining pseudonymity); nor do they show the feasibility of pseudonymous secure computation in a Bitcoin-like model.

2 Our Model

Our basic underlying model is derived from a standard setting for secure computation (namely the stand-alone model from Canetti [9]). However, our model fundamentally differs in that *we do not assume authenticated channels* between the parties. In addition, we work in a hybrid world where the parties can access a functionality that models their ability to solve a bounded number of computational puzzles per round—here, assumed for simplicity to be one. 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 (including the sender itself) at the end of that round. Honest parties cannot send a message to any 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 it can cause an honest party to receive more than $n-1$ messages in a 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 is that it may not drop or modify honest parties' messages. Thus, to recap, 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 to that party) at the end of that round.

The sequential puzzle functionality, $\mathcal{F}_{puz}$. We consider two types of cryptographic puzzles in this work. In the main body of the paper we focus on so-called sequential puzzles. In the appendix, we also consider what we call parallelizable puzzles. We refer to Section 2.1 and Appendix B for further discussion regarding the latter.

We wish to model the existence of computational puzzles that are cheap to verify but expensive (i.e., time-consuming) to solve and, in particular, can only be solved at some bounded rate. Each puzzle is tied to a specific value which must be 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 this by working in a hybrid world where there is a reactive functionality $\mathcal{F}_{puz}$ that all parties can access twice per round. The first call to $\mathcal{F}_{puz}$ models the assumption that puzzles can be solved only at a bounded rate (namely, once per round, per party). The second call represents the (idealized) assumption that verification is "free."

Formally, let λ be a (statistical) security parameter. $\mathcal{F}_{puz}$ maintains a set T (initially empty) of puzzle/solution pairs (x, h), with $x \in \{0,1\}^*$ and $h \in \{0,1\}^\lambda$; if $(x, h) \in T$ at some moment in time then we say that h *is a solution for* x. The functionality $\mathcal{F}_{puz}$ does as follows in each round:

1. Receive from each party P_i an input $(\texttt{solve}, x^{(i)})$. For $i = 1, \ldots, n$, 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 $(x_j^{(i)}, h_j^{(i)}) \in T$.

Parties may send messages to each other in between these two calls to the functionality. As in [9], we 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.

We stress that this functionality requires all parties to submit their `solve` requests at the same time. Thus, while an adversary who corrupts f parties can solve f puzzles per round, the puzzle values cannot depend on each other.

2.1 Discussion

We briefly discuss several aspects of our formal model.

Remarks on our model. We have aimed for the simplest (reasonable) model of cryptographic puzzles that captures their features of interest. Yet our model (and 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 they continue to provide the claimed guarantees.

Seemingly more worrisome is the assumption that all parties have the same computational ability, and so solve puzzles at the same rate. This, too, is not an essential feature if we simply view n as the total available "units of computational power" in the network, rather than as the total number of parties.

We make puzzle verification "free" in our model (i.e., we allow verification of an unbounded number of puzzles in each round) to prevent denial-of-service attacks in which the adversary overwhelms an honest party with (incorrect) puzzle solutions. 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 to honest parties.

Instantiation. An essential characteristic of the puzzles modeled by $\mathcal{F}_{puz}$ is that they cannot be solved any *faster* by the adversary than by a single honest process, even though the adversary can corrupt many parties and therefore solve *more* puzzles in total. This captures puzzles that require an inherently *sequential* computation to solve. A simple instantiation of a puzzle having this property is given by setting $h := H^t(x)$, where H is a hash function modeled as a random oracle and t is a parameter determining the difficulty of the puzzle. In practice, however, this puzzle is useless since verifying a solution takes as long as finding a solution. If a trusted party can be relied on to generate puzzle instances, then the puzzles considered by Rivest et al. [34] could be used; however, we do not wish to assume that such a trusted party is available.

Some cryptographic puzzles (e.g., [4, 10]) do not satisfy this property.

Recently, Mahmoody et al. [27] construct puzzles in the bounded-rate random-oracle model that come closest to realizing $\mathcal{F}_{puz}$; however, there still remains an approximation gap: in fact, an adversary may gain some constant factor of parallel speed-up with better-than-negligible probability (something not possible with $\mathcal{F}_{puz}$), and verifying a solution requires nonzero work (in contrast to $\mathcal{F}_{puz}$, where verification is free). In this work we use the $\mathcal{F}_{puz}$ model (rather than directly using a bounded-rate random-oracle) in order to abstract the properties we need and thus simplify the protocol description (and analysis). We leave for future work the problem of expanding our protocol to account for this gap; even in the worst-case, our $\mathcal{F}_{parpuz}$-based protocol applies.

Other notions of puzzles. The sequential 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.

In Appendix B, we present an ideal functionality intended to model parallelizable proof-of-work puzzles. Here, a puzzle requires some number N of computational steps to solve; however, these steps can be performed in parallel, implying that an adversary controlling f processors can solve puzzles at a f-times faster rate. (Alternatively, this can be viewed as an adversary whose processors run faster than honest processors by a factor of f.) A good example of puzzles that match this abstraction are those developed by Coelho [10].

The cryptographic puzzles used in Bitcoin are not only parallelizable, but can in fact be solved by any number of concurrent processes without communication. The probability with which a given process produces a puzzle solution is proportional to its computational power; hence, regardless of the distribution of computing resources, puzzle solutions are found 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 that even weak participants have a proportional chance of finding the next puzzle solution and thereby earning a reward.

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

We first define Interactive Set Consistency (ISC).

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

$\mathsf{corrupted\ parties}$ *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 $v_i \in V_i$.*

ISC 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 multiset), with $\vec{V}[i] = v_i$ for each honest party P_i. ISC can be viewed as a pseudonymous version of IC.

We now describe a protocol for realizing ISC in the $\mathcal{F}_{puz}$-hybrid model. First we introduce some useful terminology.

Definition 2. *A* **signed message** *is a tuple* $(\mathsf{pk}, \sigma, \mathsf{msg})$ *where σ is a valid signature on* msg *with respect to public key* pk.

If $s = (\mathsf{pk}, \sigma, \mathsf{msg})$ is a signed message, then we define $\mathsf{msg}(s) = \mathsf{msg}$ and $\mathsf{pk}(s) = \mathsf{pk}$.

Definition 3. *A* **puzzle graph** *is a tuple* $(\mathsf{sol}, \mathsf{pk}, \mathsf{children})$*, where* $\mathsf{sol} \in \{0,1\}^\lambda$ *is a puzzle solution,* $\mathsf{pk} \in \{0,1\}^*$ *is an identity string, and* $\mathsf{children}$ *is a (possibly empty) set of puzzle graphs. A puzzle graph* $g = (\mathsf{sol}, \mathsf{pk}, \mathsf{children})$ *is recursively defined to be* **valid** *if* sol *is a solution for the puzzle* $\mathsf{pk} \parallel \{\mathsf{sol}(c) \mid c \in \mathsf{children}\}$ *(where the latter are ordered lexicographically), and either* $\mathsf{children} = \emptyset$ *or else every graph in* $\mathsf{children}$ *is valid.*

If $h = (\mathsf{sol}, \mathsf{pk}, \mathsf{children})$ is a puzzle graph, then we define $\mathsf{pk}(h) = \mathsf{pk}$.

Definition 4. *We define that* **puzzle graph** h **is at depth** $\ell > 0$ **in puzzle graph** g *according to the following inductive rules:*

- *A 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.*

Definition 5. *A puzzle graph g is* **height-ℓ** *iff ℓ is the greatest integer such that g contains a depth-ℓ subgraph.*

The following lemma will be key to our protocol, as it guarantees that a puzzle graph with sufficient height must contain a subgraph generated during the first round:

Lemma 1. *If a party receives a puzzle graph g in round r, then the height of that graph is at most r. Furthermore, any depth-r subgraph in g must contain a solution output by the $\mathcal{F}_{puz}$ functionality in round* 1.

Proof. A puzzle graph contains a solution sol that must have been output by the functionality $\mathcal{F}_{puz}$ in some round; if the graph has children, then each of those children must contain a puzzle solution output by $\mathcal{F}_{puz}$ in an earlier round. From this observation, the lemma follows by induction. A height-1 graph received in round 1 must contain a single node with no children, computed in that round. A height-$(r+1)$ graph received in round $r+1$ must contain a child obtained during round r at the latest. $\square$

Intuition behind our protocol. Our protocol is inspired by the Dolev-Strong protocol for broadcast (which works by assuming a pre-established PKI) but we integrate computational resources in a nontrivial manner. In each round r, a party *accepts* a value if it has received a collection of r signatures on that value; the party then adds its own signature to the collection and relays it to all other parties. However, since in our setting we do not have a PKI and parties do not know each others' public keys, we must add an additional constraint (using the $\mathcal{F}_{puz}$ functionality) that prevents the adversary from utilizing more than one public key per corrupted party. Our constraint is based on the observation in Lemma 1, that a depth-r subgraph of a puzzle graph received in round r must have been solved in round 1. Thus in our protocol, a correct party 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. Note that the protocol only uses $f+1$ communication rounds, though we include a final "computation round" for convenience in the description. We now prove that the protocol realizes ISC.

Lemma 2. *For every honest party P_i, after round 2 it holds that $\mathsf{pk}_i \in accepted_j$ for every honest P_j.*

Proof. In the first round, each correct party P_i broadcasts its own depth-1 puzzle graph, and in the next round each correct party P_j accepts any key with a depth-1 puzzle graph. $\square$

We say that an honest party P_i *accepts* pk *in round* r if P_i adds pk to $accepted_i$ at the end of round r. We say it *accepts* pk *by round* r if it accepts pk in round $r' \leq r$.

Lemma 3. *If an honest party accepts* pk *in round $r \leq f+1$, then every honest party accepts* pk *by round $r+1$.*

An ISC Protocol for the $\mathcal{F}_{puz}$-Hybrid Model

Initially, each party P_i generates a keypair $(\mathsf{sk}_i, \mathsf{pk}'_i)$ for a digital signature scheme. We set $\mathsf{pk}_i = \mathsf{pk}'_i \| v_i$, where v_i is the input of P_i, and refer to pk_i as the *identity* of party P_i.

The subroutine $solve(\mathsf{pk}, \mathsf{children})$ returns the puzzle graph $g = (\mathsf{sol}, \mathsf{pk}, \mathsf{children})$, where sol is the solution returned from querying the $\mathcal{F}_{puz}$ functionality with $(\texttt{solve}, \mathsf{pk} \,\|\, \{\mathsf{sol}(c) \mid c \in \mathsf{children}\})$ (where the latter are in lexicographic order).

We define $\mathsf{Sign}_i(\mathsf{msg})$ to return $(\mathsf{pk}_i, \sigma, \mathsf{msg})$, where σ is a signature on msg computed using sk_i.

- **Round** 1**.** In the first round, each party P_i computes the puzzle graph $g_{i,1} = solve(\mathsf{pk}_i, \emptyset)$. It then sends $g_{i,1}$ to all parties (including to itself), and sets $accepted_i = \{\}$.
- **Round** 2 **through** $f+2$**.** For $r = 2, \ldots, f+2$, party P_i does:
  1. Set $G_{new} = \emptyset$, $S_{new} = \emptyset$.
  2. Let G be the set of valid puzzle graphs received in the previous round.
  3. Let S be the set of valid signed messages $(\mathsf{pk}, \sigma, \mathsf{msg})$ received in the previous round for which $\mathsf{pk} \in accepted_i$. ($S$ includes only one signed message per $(\mathsf{pk}, \mathsf{msg})$ pair; duplicates are discarded.)
  4. For each $g \in G$, and for each puzzle graph h that is depth $r-1$ in g, let $S_h = \{s \in S \mid \mathsf{msg}(s) = \mathsf{pk}(h)\}$. If $\mathsf{pk}(h) \notin accepted_i$ and $|S_h| \geq r-2$, then:
     (a) add $\mathsf{pk}(h)$ to $accepted_i$,
     (b) add g to G_{new},
     (c) set $S_{new} = S_{new} \cup S_h \cup \{\mathsf{Sign}_i(\mathsf{pk}(h))\}$.
  5. If $r \leq f+1$, send $g_{i,r} = solve(\mathsf{pk}_i, G_{new})$ and S_{new} to all parties.

Output the set $V_i = \{v_j \mid \mathsf{pk}'_j \| v_j \in accepted_i\}$.

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

Proof. The proof is by induction on $r \geq 2$. For the base case, when $r = 2$, observe that by Lemma 2, each honest party P_i accepts the key of every other honest party in round 2.

Suppose the lemma holds at round $r-1$ (where $r-1 \geq 2$) and suppose honest party P_i accepts pk in round r. We know that P_i must have received (in round $r-1$) at least $r-2$ signed messages s' with $\mathsf{msg}(s') = \mathsf{pk}$ and $\mathsf{pk}(s')$ accepted by P_i by round $r-1$. These $r-2$ messages must be signed using distinct public keys; furthermore, none of those public keys can be equal to pk_i because P_i only signs pk after accepting it. Applying the inductive hypothesis, every other honest party must have accepted those $r-2$ public keys by round r. From Lemma 2, we also know that every honest party has already accepted pk_i in round 2. Since P_i sends all the signed messages it has received, plus its own signature on pk, it follows that in round r every other honest party P_j receives at least $r-1$ signed messages s with $\mathsf{msg}(s) = \mathsf{pk}$ and $\mathsf{pk}(s) \in \mathit{accepted}_j$.

Next, observe that if P_i accepts pk in round r, then it must have received a puzzle 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 $g_{i,r}$ in which h is at depth r and sends $g_{i,r}$ to all other parties in round r.

It follows from both the above that every other honest party will accept pk by round $r+1$. $\square$

Lemma 4. *Each honest party accepts at most n distinct keys.*

Proof. If an honest party accepts pk in some round $r \geq 2$, then it must have received a height-$(r-1)$ puzzle graph g in the previous round. By Lemma 1, any height-$(r-1)$ graph received in round $r-1$ must contain a puzzle solved during round 1; since at most n such puzzles in total can be computed in a round, at most n distinct keys can be accepted. $\square$

Theorem 1. *Assuming secure digital signatures, for any $1 \leq f < n$ there is a polynomial-time ISC protocol in the $\mathcal{F}_{puz}$ model with $f+1$ rounds of communication, secure against f corrupted parties.*

Proof. Validity follows from Lemma 2, and boundedness follows from Lemma 4. To prove agreement, we must show that if an honest party P_i accepts pk in round $f+2$, then every honest party accepts pk by *that same round* $(f+2)$. So, say P_i accepts pk in round $f+2$. Then P_i must have received f signatures on pk from distinct previously-accepted keys. Observe that among these f previously-accepted keys, at least 1 must belong to some honest party P_j: by Lemma 2, at least $n-f$ keys from correct parties are accepted by P_i in round 2, and by Lemma 4 at most n keys in total are accepted among all the correct parties. Since a correct party only signs pk after accepting it, P_j must have accepted pk in round $f+1$ or earlier. Therefore, by Lemma 3, every correct party accepts pk in round $f+2$ or earlier. $\square$

Message Complexity. In the worst case, each party 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 party accepts at most n keys, it publishes at most $O(\lambda n^2 f)$ message bits. As a further optimization, each party 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 party.

3.1 A Lower Bound on the Round Complexity

Our ISC protocol is round-optimal for *deterministic*[2] protocols:

Theorem 2. *Say $n - f \geq 2$. (Otherwise, ISC is trivial.) Any deterministic ISC protocol in the $\mathcal{F}_{puz}$-hybrid model that tolerates f faults must have at least $f + 1$ rounds of communication.*

A proof of the above is given in Appendix A. The proof is inspired by the lower bound on the round complexity of broadcast with a PKI [13, 1], but requires several modifications.

4 Pseudonymous MPC from Sequential Puzzles

In the previous section, we showed a protocol for ISC in the $\mathcal{F}_{puz}$-hybrid model. This protocol allows the participating parties to agree on a set of public keys, with honest parties' public keys guaranteed to be included; thus, this effectively bootstraps a (pseudonymous) PKI.

In this section, we show that ISC can be used to achieve (pseudonymous) secure computation of general functionalities. We first show that the standard Dolev-Strong algorithm [13] can be run following ISC in order to implement a pseudonymous version of *secure broadcast*; this, in turn, can be used as part of standard protocols for secure multi-party computation (e.g., the GMW protocol) [20]) to implement pseudonymous secure computation. In general, any other multi-party computation protocol (such as BGW [6], for example) could be used in place of BGW; the fairness and resilience properties would carry over from such protocol, since our ISC protocol provides the best possible resilience ($f < n$) and fairness condition (guaranteed termination).

[2]Technically, our protocol is not deterministic because of the initial key-generation step. Our proof can be extended to apply to deterministic protocols given access to an ideal functionality corresponding to a digital signature scheme.

4.1 Definition of Pseudonymous Security

We motivate the need to define a relaxed, *pseudonymous* notion of security by pointing out that the standard notion of security [9] cannot be achieved in a setting where the underlying network does not offer authenticated communication. Intuitively, the issue is that there is nothing distinguishing the parties from each other; thus, it is not even clear which of the n parties should play the role of P_1, which the role of P_2, etc. As a consequence, we must either restrict attention to functions that treat each of their n inputs *symmetrically*, or else we must allow the attacker to arbitrarily choose the order in which the parties' n inputs are entered into the computation. We choose the second approach, which is in fact more general than the first.

The relaxed notion of pseudonymous secure computation that we consider may still be useful in many circumstances. First, many functions of interest (such as majority, average, etc.) are invariant under permutations of the input values, and are therefore unaffected by the relaxation. Also, in many scenarios the ordering of the inputs may be irrelevant; for example, in a game of online poker against anonymous opponents, each party might only be concerned with having the game played honestly and correctly.

The formal definition. To define pseudonymous secure computation, we define an appropriate ideal world which is a relaxation of the one normally considered when defining secure computation. (We have already defined our real world in Section 2, and as usual [9] we will define a real-world protocol to be secure if the actions of any adversary attacking the protocol can be simulated by an adversary in the ideal world.) The main relaxation is that in pseudonymous MPC *we allow the adversary to choose an arbitrary permutation on the inputs.*

Our execution model is formally defined in Figure 2, but we describe the main points below: Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a function to evaluate. Each party P_i holds input $x_i \in \{0,1\}^\lambda$, where λ is the security parameter. The ideal-world adversary $\mathcal{S}$ is an interactive Turing machine that controls a set $C \subset \{1, \ldots, n\}$ of corrupted parties, and we denote the set of uncorrupted parties by $I = [n] \setminus C$. The adversary $\mathcal{S}$ receives the inputs of the corrupted parties, as well as an auxiliary input z. Execution in the ideal world proceeds as follows:

- **Input substitution:** Each honest party $P_i \in I$ sends its input x_i to the trusted party evaluating f. The adversary $\mathcal{S}$ sends arbitrary inputs x_i on behalf of each corrupted party $P_i \in C$. Let $\vec{x} = (x_1, \ldots, x_n)$

denote the input values sent by the n parties.[3]

- **Permutation:** $\mathcal{S}$ chooses a permutation π on $\{1, \ldots, n\}$.
- **Computation:** Let $\vec{x}'$ be the vector obtained by applying the permutation π to the entries of $\vec{x}$; i.e., $x'_{\pi(i)} = x_i$. The trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$, and returns $y_{\pi(i)}$ to party P_i.
- **Output:** Each honest P_i outputs the value given to it by the trusted party. In addition, $\mathcal{S}$ outputs an arbitrary function of its view.

4.2 Protocol for Pseudonymous Multi-party Computation

The GMW protocol for multi-party computation makes use of two communication primitives: private authenticated channels, and broadcast. We will need to emulate (the pseudonymous version of) these in the $\mathcal{F}_{puz}$-hybrid model. The basic idea is for each party to generate a public encryption/signing keypair, and then run the ISC protocol so that all the parties agree on a set of public keys. This effectivey amounts to bootstrapping a pseudonymous PKI that includes a pseudonym for each honest party and at most f pseudonyms for the adversary. Thereafter, standard encryption and signing techniques can be used to realize (pseudonymous) authenticated and private channels between the parties; the Dolev-Strong algorithm can be used to give pseudonymous broadcast.

In more detail, the protocol for evaluating a function f proceeds as follows:

1. First, each party P_i generates a signing and encryption keypair $\mathsf{pk}_i, \mathsf{sk}_i$. 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 set of public keys which it can order lexicographically to give a vector $(\mathsf{pk}'_1, \ldots, \mathsf{pk}'_n)$. We remark that this effectively defines a permutation of the parties, and that each party can determine its permuted index.
2. Next, each party P_i executes the GMW protocol [20]. Note that the GMW 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 PKI. When the protocol calls for a private message, the signing and encryption keys are used as described above.

[3]We assume parties have identifiers in the ideal world, but these identifiers are only used as a formalism to allow $\mathcal{S}$ to address the corrupted parties; the identifiers have no inherent meaning.

Theorem 3 (Pseudonymously Secure SFE)**.** *In the $\mathcal{F}_{puz}$-hybrid model, we achieve SFE for general 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 an arbitrary number of corruptions, we ensure security with unanimous abort.*

Proof. First we prove that the protocol described above satisfies the case for honest majority. Since the underlying GMW 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 GMW 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.

Finally in the case without honest majority (i.e., $n > f > \lfloor n/2 \rfloor$) the GMW [20] protocol guarantees privacy but not fairness. $\square$

5 Conclusion 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 widely held belief that nothing interesting can be achieved in networks without authenticated channels or a PKI, we show that by placing a strict bound on each party's computational resources it is possible set up a (pseudonymous) PKI and achieve (pseudonymous) notions of broadcast and secure computation. Although our work does not directly apply to the actual Bitcoin protocol, our results show that distributed-computation models resembling Bitcoin can lead to rich applications with non-trivial security guarantees.

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 communication 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 [16, 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] Marcin Andrychowicz and Stefan Dziembowski. Distributed cryptography based on proofs of work. Cryptology ePrint Archive, Report 2014/796, 2014.

[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] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology—Crypto 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] Shlomi Dolev, Nova Fandina, and Ximing Li. Nested merkles puzzles against sampling attacks. In Mirosaw Kutyowski and Moti Yung, editors, *Information Security and Cryptology*, volume 7763 of *Lecture Notes in Computer Science*, pages 157–174. Springer Berlin Heidelberg, 2013.

[15] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology—Crypto '92*, volume 740 of *LNCS*, pages 139–147. Springer, 1993.

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

[17] 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 *LNCS*, pages 482–501. Springer, 2002.

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

[19] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. Eurocrypt 2015, to appear.

[20] 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 (STOC)*, pages 218–229. ACM Press, 1987.

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

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

[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] 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.

[28] Ralph C Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[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 LaViola, Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for Bitcoin. Technical Report CS-TR-14-01, University of Central Florida, 2014. Availale at `http://tr.eecs.ucf.edu/78`.

[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 Rivest, Adi Shamir, and David 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 [13].

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. *Say $n - f \geq 2$. (Otherwise, ISC is trivial.) Any deterministic ISC protocol in the $\mathcal{F}_{puz}$-hybrid model that tolerates f faults must have at least $f + 1$ rounds of communication.*

Proof. 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. [4] 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

[4]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 [13]. We have chosen our approach because it allows us to use a simple and concrete representation of the model for presenting our positive results.

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 5. *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. $\square$

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 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 [13, 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 6. *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 Parallelizable Proof-of-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 3.

Lemma 7. *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 8. *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 7 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 3).

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 9. *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. $\square$

Theorem 4. *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 1, substituting Lemmas 7,8, and 9 for Lemmas 2,3, and 4. $\square$

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

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