# Fair Multiparty Coin Tossing from Minimal Assumptions

Marshall Ball
New York University
marshall.ball@cs.nyu.edu

Miranda Christ
Columbia University
mchrist@cs.columbia.edu

Yevgeniy Dodis
New York University
dodis@cs.nyu.edu

Rachit Garg
New York University
rg5134@nyu.edu

**Abstract**

Coin flipping in the presence of a dishonest majority is a fundamental cryptographic primitive whose requirements lack a clean characterization. Recent work (Bonneau et al., Eurocrypt 2025) showed a lower bound that fair dishonest-majority coin-flipping implies delay functions. However, until now known upper bounds exhibited a significant gap: All existing protocols rely on assumptions that we do not know how to instantiate in the plain model.

In this work, we close this gap. Specifically, we show that fair $n$-party coin flipping in the presence of up to $n-1$ malicious corruptions follows from the *minimal* assumption of delay functions. This completes the equivalence between delay functions and fair dishonest-majority coin-flipping protocols.

## 1 Introduction

Multiparty coin tossing[1], originally introduced by Blum [Blu82], is a fundamental cryptographic primitive that enables mutually distrustful parties to agree on a shared coin. For many applications, it is crucial that this coin is *fair*: unpredictable and unbiasable by any coalition of parties. However, shortly after this problem's introduction Cleve [Cle86] showed in a seminal paper that tossing a fair coin is impossible if a malicious adversary can corrupt half the parties. A beautiful line of work has studied exactly how biasable such coin-tossing protocols are, achieving optimal tradeoffs between round complexity and bias.

Years later, Goldschlag and Stubblebine [GS98], and Boneh and Naor [BN00], observed that *timed cryptography* gives a clever way around Cleve's impossibility. In particular, Boneh and Naor showed that a novel notion of *timed commitments* indeed enabled a two-party coin tossing protocol with negligible bias. Inspired by prior work on "timed cryptography" [DN93, RSW96, BG97], a timed commitment is a commitment that has a fine-grained notion of hiding: adversaries running in a *short* amount of time learn nothing (even with polynomial parallel processing), whereas it is possible to "force open" the committed value in a *moderately long* amount of time.

Since these seminal works, there has been an explosion of work constructing general fair multiparty coin tossing from "time-based" crypto. However, the security of all of these protocols relies on assumptions which we do not know how to provably instantiate outside of the random oracle model (see Section 1.2).

[1] We use coin tossing and coin flipping interchangeably.

Thus, to our knowledge, the following elementary question remains open over 40 years after Blum introduced the idea of coin tossing:

> *Can we construct a fair dishonest majority n-party coin toss from standard assumptions?*

Recent work by Bonneau et al. [BBCE25] showed that *delay functions* are necessary for fair multiparty coin-flipping. [2] Loosely, a delay function, introduced by Abram et al. [ARS24] and a weakening of the *verifiable delay functions* of [BBBF18], is a function which can be evaluated in a moderate amount of time, but is infeasible to evaluate quickly even with high parallelism. (An informal definition can be found Section 1.1, or see Definition 1 for a complete formalization.) We note that this notion is not even known to imply one-way functions, and black-box constructions have been shown impossible [GM24].

This leads us to a refinement of our previous question:

> Can we construct a fair dishonest-majority $n$-party coin toss from the *minimal assumption* of delay functions?

In this work, we give an affirmative answer to this question.

## 1.1 Our Results

We first informally recall the notion of a delay function we consider. A $t(\lambda)$-delay function is a tuple of p.p.t. algorithms $(\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ together describing a function that is hard to compute in fewer than $t(\lambda)$ sequential steps. $\mathsf{Setup}$ outputs some public parameters $\mathsf{pp}$. $\mathsf{SampleInput}$ takes in the public parameters, and samples an input $x$ from a distribution over which $\mathsf{Eval}$ is hard to compute. Finally, what one thinks of as the actual function is $\mathsf{Eval}$, a deterministic algorithm that takes in $(\mathsf{pp}, x)$ and outputs $y$.

The delay function should be *sequential*, in that (a bit informally) for any $\mathcal{A}$ running in fewer than $t(\lambda)$ sequential steps,

$$\Pr_{\substack{\mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda)\\ x\leftarrow\mathsf{SampleInput}(\mathsf{pp})}}[y' = \mathsf{Eval}(\mathsf{pp}, x) : y' \leftarrow \mathcal{A}(\mathsf{pp}, x)] \leq \mathsf{negl}(\lambda).$$

This is a simplified and slightly weaker version of the full definition (Definition 2), where the adversary is also allowed sequentially-unbounded preprocessing on $\mathsf{pp}$.

This notion of a delay function is essentially the weakest notion one could imagine, and far weaker than VDFs or time-lock puzzles which require fast verification or fast sampling of $(x, y)$ pairs. We refer the reader to [BBCE25] for further discussion of this definition compared to others.

Somewhat surprisingly, we show that this weak object is sufficient to construct $n$-party coin-flipping protocols that are fair even if $n - 1$ parties are corrupted. That is, we show the following:

**Theorem 3, informal.** Using *only* a $t(\cdot)$-sequential delay function, we can construct an $n$-party coin-flipping protocol that is fair under a dishonest majority, provided that the adversary runs in $O(t(\lambda))$ sequential steps.

Along the way, we use a delay function to construct several other tools of interest. In doing so, we develop techniques for using delay functions, which we believe are of independent interest.

[2] They additionally showed that delay functions are sufficient for fair coin flipping in the random oracle model.

## 1.2 Related Work

**Coin Flipping and Randomness Beacons.** The problem of generating shared randomness dates back to the early days of cryptography. Blum [Blu82] first formalized two-party coin flipping as a protocol for two mutually untrusting parties to agree on a random bit. This primitive was quickly recognized as a crucial building block for larger protocols, such as mental poker and fair exchange. However, Cleve's seminal impossibility result [Cle86] demonstrated that no coin-flipping protocol can achieve perfect fairness if a majority of participants are dishonest, establishing a fundamental limitation. The most common approach, the commit-reveal paradigm, is vulnerable to a "last-revealer" attack, where a malicious party aborts after seeing honest parties' inputs. To overcome this in the honest-majority setting, protocols have incorporated techniques like Verifiable Secret Sharing and later Publicly Verifiable Secret Sharing (PVSS) to enable recovery of an aborting party's contribution. Some modern distributed randomness beacons (DRBs) build on these ideas, but fundamentally rely on an honest majority for security. We refer the reader to [CMB23] for a thorough discussion of such protocols.

Prior to this work, the only fair $n$-party coin-flipping protocol in the plain model was due to Freitag et al. [FKPS21][3]. They show that a primitive called a *non-malleable time-lock puzzle* suffices. However, they assume various subexponential cryptographic security assumptions, including *keyless* multicollision resistant hash functions [BKP18]. At present, we do not have a provably secure instantiation of this primitive from *any* standard assumption. Bitansky et al. [BKP18], who introduced the assumption, suggested keying asymptotic variants of SHA2 with the first $n$ digits of $\pi$. Proving this secure from any reasonable assumption is well outside of our current techniques.

**Time-based Cryptography.** The idea of leveraging computational time as a cryptographic resource provided a path to circumventing Cleve's impossibility. Early work on intentionally slow computations includes time-lock puzzles by Rivest, Shamir, and Wagner [RSW96], which introduced the concept of encrypting a message such that it could only be decrypted after a predetermined amount of sequential computation. Boneh and Naor [BN00] built on this by introducing *timed commitments*, a primitive that allows a party to commit to a value that remains hidden for a specific duration, after which anyone can compute the value. They explicitly noted coin flipping as a key application. This line of work culminated in the formalization of Verifiable Delay Functions (VDFs) by Boneh et al. [BBBF18], which require a specific amount of sequential time to compute but can be verified efficiently. VDFs have become the central tool in modern dishonest-majority beacon protocols, such as Unicorn [LW17], which uses a delay function to prevent an adversary from biasing the output even if it sees all other contributions before making its own. Several subsequent schemes [CATB23, CCB24, LHW+22, SJH+23, ARS24] augment the Unicorn blueprint to improve efficiency, but all rely on a random oracle or another idealized model.

# 2 Technical Overview

## 2.1 Background: dishonest-majority coin-flipping protocols

**Dishonest-majority coin-flipping protocols.** In a coin-flipping protocol, $n$ parties interact to generate a random output $y \in \{0, 1\}$. This protocol should satisfy both:

[3]Though [MT19] claims to construct such a protocol from homomorphic time-lock commitments, Freitag et al. observe that this protocol is biasable by a malicious adversary due to malleability issues.

***Unbiasability*:** $y$ is computationally indistinguishable from uniform, even in the presence of corrupted parties deviating from the protocol in an attempt to influence its output.

***Agreement*:** Honest parties should always output the same $y$.

In a seminal work, Cleve showed that it is impossible to construct coin-flipping protocols in the presence of a dishonest majority. He showed a simple attack that introduces non-negligible bias to any coin-flipping protocol that satisfies agreement. His attack essentially involves an adversary that asks in each round, "what would the output be if I aborted now versus later?" and acts accordingly.

Later work observed that this attack can be avoided if this abort-now-or-later function *is too slow to compute during the protocol*. By the time the adversary decides whether to abort, the protocol has terminated. This led to the aforementioned line of work building coin-flipping protocols using *delay functions*. Formally, these protocols circumvent Cleve's impossibility by achieving unbiasability only against adversaries that are fine-grained in some sense, and unable to compute the delay function during the protocol.

**The Unicorn protocol.** An elegant approach to constructing dishonest-majority coin-flipping protocols using delay functions was formalized by Lenstra and Wesolowski [LW17]. Their protocol, Unicorn, uses a delay function $f$, where predicting $f(x)$ on a random $x$ with non-negligible success requires at least $t$ sequential steps. It also uses a random oracle $H$. We present a slight variation of Unicorn below.[4]

The protocol commences at some time $T_0$ and ends at time $T_1$, such that it is infeasible to perform $t$ sequential steps during that time interval. Between $T_0$ and $T_1$, each party $P_i$ for $i \in [n]$ posts a random value $r_i \leftarrow\!\!\$\ \{0,1\}^\lambda$ on a public bulletin board. The output of the protocol is $y = H(f(r_1)||\ldots||f(r_n))$, which can be computed in a longer offline phase after $T_1$.

The output of the Unicorn protocol is pseudorandom as long as at least *one* party is honest. Observe that if even one party contributes a truly random $r_i$, $f(r_i)$ is unpredictable before time $T_1$. Therefore, the adversary cannot query $f(r_1)||\ldots||f(r_n)$ to the random oracle. It thus cannot distinguish $y$ from uniform before $T_1$.

**Other delay-based protocols.** Subsequent works [CATB23, CCB24, LHW+22, SJH+23, ARS24] have constructed delay-based coin-flipping protocols with greater efficiency, or using different cryptographic primitives. Most are similar in spirit to Unicorn, and involve parties individually contributing randomness that is somehow combined using a delay function, to eventually yield a single output.

Most follow a similar paradigm, where the interactive phase of the protocol is time-bounded. Parties may be allowed longer preprocessing and postprocessing phases, before and after the interactive phase.

**Coin flipping from weaker assumptions.** Bonneau et al. [BBCE25] showed that dishonest-majority coin-flipping protocols require a fairly weak notion of a delay function.[5] This requirement holds even for protocols that allow preprocessing and postprocessing phases. Somewhat surprisingly, they show that such delay functions are also *sufficient* for constructing dishonest-majority coin-flipping protocols *in the random oracle model*. However, the use of a random oracle leaves a gap between their upper and lower bounds.

A natural first question is whether one could relax the random oracle to a one-way function. But could we be even more ambitious? A truly minimal protocol would rely *only* on delay functions, which do not imply one-way functions.[6] However, this task seems challenging and maybe impossible, as the delay function of [BBCE25] is a very weak object: It is simply a function that cannot be computed in bounded depth. As we'll soon see, removing the random oracle seems to call for tools such as commitments and

[4] In Unicorn, the output is $H(f(H(r_1||\ldots||r_n)))$. Our modification makes our later removal of the random oracle slightly cleaner.
[5] We use this notion in our work, and a formal definition appears in Section 3.
[6] Formally, delay functions and one-way functions are incomparable; however, delay functions appear weaker in spirit.

zero-knowledge proofs. It is unclear whether delay functions can realize the properties provided by these tools.

**The purpose of the random oracle.** Our first step in removing the random oracle is understanding why we needed it in the first place. We'll use the Unicorn protocol as a case study.

Recall that in Unicorn, the output is computed as $y = H(f(r_1)||\dots||f(r_n))$. Consider the case where only party $i$ is honest. Then from the perspective of the adversary, $f(r_j)$ for all $j \neq i$ are known values, and $f(r_i)$ is unpredictable. More precisely, $f(r_i)$ has *computational min-entropy*: for any guess $y'$ of the adversary given $r_i$, $\Pr[f(r_i) = y'] \leq \mathsf{negl}(\lambda)$. An application of the random oracle allows us to transform this computational min-entropy source $f(r_1)||\dots||f(r_n)$ into uniform randomness.

One could attempt to use randomness extractors. However, extractors from min-entropy sources require a short uniformly random seed that is independent from the source. In other words, they achieve no meaningful guarantee if the source depends on the seed. In our setting, the seed must be public knowledge as it is required to compute the protocol output. But this means the source $(f(r_1)||\dots||f(r_n))$ is chosen partly by an adversary that knows the seed. Therefore, randomness extractors are not useful here.

## 2.2 Removing the random oracle

**A first attempt: making $f$ pseudorandom.** If $f(r_i)$ were *pseudorandom* rather than just unpredictable, we would be in a better situation. That is, we want the adversary's advantage in the following distinguishing experiment to be negligible: At time $T_0$, the adversary is given either $(r_i, f(r_i))$ or $(r_i, u)$ where $r_i$ and $u$ are uniform. The adversary must output its guess at time $T_1$. We now consider pseudorandom delay functions $f$ outputting a single bit.

Once $f$ is pseudorandom, we could hope to compute the output as $\bigoplus_{j\in[n]} f(r_j)$. While intuition might suggest that this output is pseudorandom, there is a subtle issue. The pseudorandomness property above says nothing about the adversary's ability to quickly find $r_j$ whose output under $f$ is correlated with $f(r_i)$. For example, a (very annoying) delay function $f$ could have the property that $f(x) \oplus f(\bar{x}) = 0$, where $\bar{x}$ denotes the complement of $x$. If this bad $f$ were used, the adversary could always induce an output of 0 as follows:[7]

Upon seeing the honest party's contribution $r_i$, submit $\bar{r_i}$ as one corrupted party's contribution. For all other corrupted parties, submit the same arbitrarily chosen input $x$. The output of the protocol will be

$$\left(\bigoplus_{j=1}^{n-2} f(x)\right) \oplus (f(r_i) \oplus f(\bar{r_i})) = 0.$$

We call this a *malleability issue*—the adversary can choose contributions $r_j$ such that $f(r_j)$ is *related to* $f(r_i)$.

Before continuing, we remark that we actually *can* make $f$ pseudorandom without much difficulty. We do so in Section 3.2, using Goldreich-Levin [GL89] and an observation from [AMZ24] that the Goldreich-Levin reduction is low-depth.

**Solving the malleability issue: two parties.** For a protocol with only two parties, there is a simple solution to the malleability issue, introduced by Boneh and Naor [BN00]. Boneh and Naor used timed commitments, which they built from the iterated squaring assumption; the observation that it can be instantiated only from the minimal notion of a delay function is ours. We sketch the protocol here, to build intuition for our full protocol.

[7] In protocols with an even number of parties.

This protocol will make use of *delay commitments*: $c \leftarrow \mathsf{Com}(m; r)$ is perfectly binding and hiding against bounded-depth adversaries. This allows parties to send commitments that are hiding during the interactive phase of the protocol. Our commitments will have the additional property (which is not necessary for the two-party protocol) that anyone can open the commitments, without interaction or knowledge of $r$, in the slow postprocessing phase. We show how to construct such commitments from pseudorandom delay functions in Section 3.5. The construction is quite simple: a commitment to $m$ is $(r, f(r) \oplus m)$.

For the two-party protocol we will require some complexity leveraging. If the delay function is computable in $t$ sequential steps, we will require that the delay commitments are hiding against adversaries allowed to perform $T \gg t$ sequential steps. We're now ready to see the two-party protocol; call the two parties Alice and Bob. For simplicity, assume that $f$ outputs a single bit.

1. Alice samples $x_A \leftarrow \{0,1\}^\lambda$ and sends Bob $c = \mathsf{Com}(x_A; r)$.
2. Bob samples $x_B \leftarrow \mathcal{U}_\lambda$ and sends $x_B$.
3. Alice opens $c$, sending $(x_A, r)$ to Bob.
4. Alice and Bob both compute and output $f(x_A) \oplus f(x_B)$, if all messages were sent. Otherwise if Bob aborts, Alice outputs $f(x_A)$. Similarly if Alice aborts, Bob outputs $f(x_B)$.

First, suppose Alice is malicious and causes the protocol to output 1 with probability significantly greater than $1/2$. Then consider the following adversary breaking pseudorandomness of the delay function. It first simulates Alice to sample $x_A$ in its preprocessing phase and precomputes $f(x_A)$, then simulates Alice to generate the commitment. It then receives $x_B$ from its challenger and simulates Alice to decide whether to open $c$. If Alice does not open $c$, it guesses that $f(x_B)$ is equal to $f(x_A) \oplus 1$. If Alice does open, it guesses that $f(x_B)$ is equal to 1. Since the output is equal to 1 with probability significantly greater than $1/2$, Alice guesses $f(x_B)$ correctly with non-negligible advantage.

Next, suppose Bob is malicious and causes the protocol to output 1 with probability significantly greater than $1/2$. Skipping the simple case where Bob aborts, this means that Bob outputs $x_B$ such that $f(x_A) \oplus f(x_B) = 1$ with probability greater than $1/2$. Therefore, we can use Bob to construct an adversary that breaks hiding of the commitment, distinguishing between messages $m_0$ and $m_1$ evaluating to 0 and 1 under $f$. This adversary must compute $f$, which is why we required complexity leveraging.

Observe that there was no malleability issue here because Bob had to output $x_B$ *in the clear*. However, if we tried to add a third party Charlie, he would see Bob's input in the clear and the malleability issue would return; he could submit $\bar{x_B}$ as before. If we tried to fix this by making Bob send a commitment to $x_B$, he could in principle maul Alice's commitment to commit to $\bar{x_A}$. We appear to be at a roadblock.

**Solving the malleability issue: $n$ parties.** As we just saw, we cannot extend the protocol to $n$ parties by having every party send a commitment, as Charlie can commit to a function of $x_A$ without violating hiding. A natural fix is to make every party submit a commitment $c_i$ along with a zero-knowledge proof of knowledge that they know $v_i, r_i$ such that $c_i = \mathsf{Com}(v_i; r_i)$. The commitments are then opened in the slow postprocessing phase; the output is the XOR of all $f(v_i)$ whose corresponding proofs of knowledge were valid.

Intuitively, this protocol should be unbiasable as an adversary biasing the protocol towards 1 would be able to produce $v_i$'s such that

$$\Pr\left[\left(\bigoplus_{j \text{ honest}} f(v_j)\right) \oplus \left(\bigoplus_{i \text{ corrupted}} f(v_i)\right) = 1\right] \geq \frac{1}{2} + \epsilon$$

for some non-negligible $\epsilon$. Therefore, $\left(\bigoplus_{i \text{ corrupted}} f(v_i)\right) \oplus 1$ would be a good predictor of $\left(\bigoplus_{j \text{ honest}} f(v_j)\right)$. However, the commitments should hide $\bigoplus_{j \text{ honest}} f(v_j)$.

While this idea indeed works, there are two immediate challenges to instantiating it: (1) we do not know how to build zero-knowledge proofs of knowledge from delay functions, and (2) our delay commitments are hiding only against bounded-depth adversaries.

The importance of (1) should be self-evident. (2) is a bit more subtle. When proving security of the protocol, we'll first show that you can replace honest parties with simulators, and replace corrupted parties' $v_i$'s with values derived from a knowledge extractor, without changing the output of the protocol. Therefore if the original protocol was biased, this simulator-extractor-hybrid protocol will be biased as well. We'll then want to argue that if the output of this hybrid protocol is biased, an adversary can break hiding of the commitment. However, this adversary will need to run the simulator and extractor, and hiding only holds against bounded-depth adversaries. Therefore, in order to reach a contradiction it is crucial that this adversary is—and therefore the simulator and knowledge extractor are—bounded-depth.

A second set of challenges arises from the fact that all parties participate in verification and must agree on its outcome. That is, consider a corrupted party $P_i$ proving knowledge of an opening to its commitment. It may send a valid proof to $P_j$ and an invalid proof to $P_{j'}$, causing $P_j$ and $P_{j'}$ to have different outputs. A natural fix is for $P_j$ to check all proofs on behalf of *all other parties*, and include $P_i$'s contribution only if *all* proofs are valid. However, defining a notion of "public verification" that enables these checks requires some care to prevent further attacks. First, public verification should agree with standard verification. Second, and more subtly, a malicious $P_{j'}$ should not be able to craft a challenge that an honest prover cannot answer satisfactorily. This would allow even a single corrupted verifier to block all honest contributions by sending bad challenges. Our solution is the definition of *robust public verification* given in Section 3.6.

**Hamiltonian cycle interactive proof.** We observe that we can carefully adapt Blum's classic zero-knowledge proof for the language of Hamiltonian cycles. In this protocol, the prover commits to every entry in the adjacency matrix of a random permutation of the given graph. It receives a random challenge $\alpha \in \{0, 1\}$. If $\alpha = 0$, it opens all commitments and sends the permutation. If $\alpha = 1$, it opens only the commitments along the Hamiltonian cycle. This protocol is zero-knowledge, and the simulator generates and opens appropriate commitments. It is also a proof of knowledge: if the prover can respond satisfactorily to $\alpha = 0$ and $\alpha = 1$, the extractor obtains both the permutation and permuted cycle. It then reverses the permutation to find the cycle in the original graph.

Observe that the simulator must generate commitments, which involves evaluating the delay function, so it cannot be low depth. However, we observe that it suffices for our application that the simulator is low-depth *once it receives the graph*; it may use a long preprocessing phase to generate materials for the commitments. We also show that the extractor can be run in low depth.

**Broadcast.** We have not yet mentioned a key ingredient in prior work. In order to ensure that honest parties agree on the output of the protocol, most works assume broadcast channels. As broadcast can be realized using digital signatures and Dolev-Strong [DS83], this is a reasonable assumption in their settings, which assume one-way functions.

However, in our minimal setting, we must construct a broadcast protocol using only delay functions. We do so by building one-time signatures, which we use in Dolev-Strong (Section 3.7).

## 2.3 Final protocol

In summary, our $n$-party protocol works as follows:

1. Each party $P_i$ broadcasts a commitment $c_i$ to a random contribution $r_i$.

2. Each party $P_i$ proves in zero-knowledge to every party $P_j$ that they know an opening of $c_i$.

3. Each party checks all proofs on behalf of all other parties. If *all* of $P_i$'s proofs were accepted, its contribution is deemed valid.

4. The output is the XOR of all valid random contributions $r_i$, which are recovered by slow-opening the commitments $c_i$.

The proof of knowledge ensures that every party *knows* their random contribution, solving any malleability issues. Zero-knowledge and hiding of the commitments ensure that the corrupted parties choose their contributions independently of the honest parties' contributions. Therefore, the protocol is unbiasable.

Robust public verification of the zero-knowledge protocol ensures that all honest parties agree on which contributions are valid.

**Efficiency.** It is crucial for our security proof that several algorithms (e.g., the knowledge extractor and the public verification algorithm) can be run in few sequential steps. Doing so requires a large amount of parallelism, linear in the delay parameter. However, we emphasize that it suffices that there *exist* such low-depth algorithms for the reduction; no parties need to *run* these algorithm in low depth in the protocol. Therefore, parties can participate in the protocol even if they have only a single processor.

# 3 Preliminaries

Let $\lambda$ denote the security parameter. We write $\mathsf{poly}(\lambda)$ to mean a polynomial function in $\lambda$. We say a function $f$ of $\lambda$ is *negligible* if $f(\lambda) = O(\frac{1}{\mathsf{poly}(\lambda)})$ for every polynomial $\mathsf{poly}(\cdot)$, and we write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to mean that $f$ is negligible. We say that a probability is *overwhelming* if it is at least $1 - f(\lambda)$ for some negligible function $f$. We write "p.p.t." to mean probabilistic polynomial time. We use $[n]$ to denote the set $\{1, \ldots, n\}$.

**Sequential and parallel time.** We use the notions of sequential and parallel time considered in [BBBF18]. The *total* time of an algorithm is the time required to run on a machine with a single processor. The *parallel* time of an algorithm with $p$ processors is the total number of sequential steps required to run this algorithm on a PRAM machine with $p$ processors. We allow our PRAM machines to compute AND and OR functions of polynomial fan-in in constant time. We also allow our PRAM machines to take as input very (though still polynomially) long strings and access any location in a single operation. When we refer to parallel time but leave the number of processors unspecified, we mean the parallel time of an algorithm with $\mathsf{poly}(\lambda)$ processors. As shorthand, we sometimes say an algorithm running in parallel time $t$ has *depth* $t$.

## 3.1 Delay Functions

We recall the definition of a delay function from [BBCE25]. Although there are many other variants in the literature (e.g., [BBBF18, ARS24]), this notion of a delay function is implied by dishonest-majority coin-flipping protocols.

**Definition 1** (Delay function ([BBCE25]))**.** A delay function is a tuple of algorithms $(\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ such that:

$\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$**:** Setup takes as input the security parameter and outputs public parameters pp.

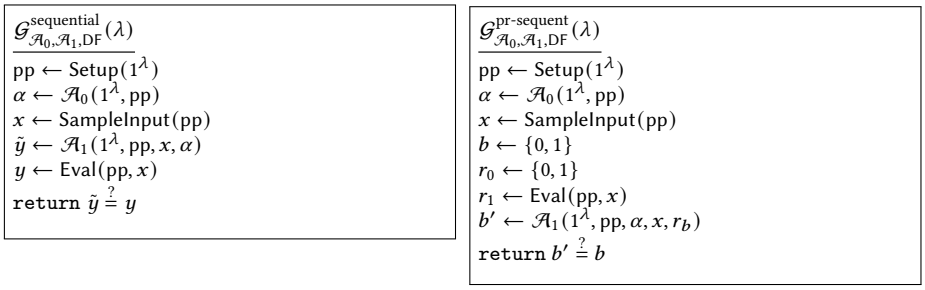



Figure 1: Delay function sequentiality and pseudorandomness games

$\mathsf{SampleInput}(\mathsf{pp}) \to x$**:** SampleInput is a randomized algorithm that takes as input the security parameter and outputs a random value $x$ from the input distribution.[8]

$\mathsf{Eval}(\mathsf{pp}, x) \to y$**:** Eval is a function that takes an input $x$ and outputs a corresponding evaluation $y$.

The delay function must satisfy *efficiency* and *sequentiality*:

**Efficiency.** SampleInput and Eval must be p.p.t. computable algorithms.

**Sequentiality.** For some delay parameter $t(\lambda)$, evaluation must require $\omega(t)$ sequential steps. More precisely, Eval must be $t(\lambda)$-sequential, as defined in [BBBF18] and recalled below.

**Definition 2** ($t(\cdot)$-sequentiality ([BBBF18]).)**.** A delay function $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ is $t(\cdot)$-sequential if for all randomized algorithms $\mathcal{A}_0, \mathcal{A}_1$ running in total time $\mathsf{poly}(\lambda)$, and $\mathcal{A}_1$ running in parallel time at most $O(t(\lambda))$[9],

$$\Pr\left[\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,\mathsf{DF}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Sequentiality requires only that outputs of Eval are hard to compute in few sequential steps. A stronger notion is a *pseudorandom delay function*, where outputs of Eval on random inputs appear random to sequentially bounded adversaries.

**Definition 3** (Pseudorandom delay function)**.** A pseudorandom $t(\cdot)$-delay function is a delay function that additionally satisfies $t(\cdot)$-sequential pseudorandomness, defined below (Definition 4).

**Definition 4** ($t(\cdot)$-sequential pseudorandomness)**.** A delay function $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ is $t(\cdot)$-sequentially pseudorandom if for all randomized algorithms $\mathcal{A}_0, \mathcal{A}_1$ running in total time $\mathsf{poly}(\lambda)$, and $\mathcal{A}_1$ running in parallel time at most $O(t(\lambda))$,

$$\left|\Pr\left[\mathcal{G}^{\text{pr-sequent}}_{\mathcal{A}_0,\mathcal{A}_1,\mathsf{DF}}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

[8] We emphasize that this distribution may not be uniform.

[9] While [BBBF18] restrict the adversary to run in at most exactly $t(\lambda)$ steps, we allow it to run in $O(t(\lambda))$ steps allowing us to drop lower-order terms later on.

## 3.2 Pseudorandom Delay Functions

We observe that any delay function can be used to construct a pseudorandom delay function. This is an immediate corollary of the Goldreich Levin Theorem, where the Goldreich Levin Algorithm can be written as a randomized algorithm that is depth preserving. This observation was made by [AMZ24] in the context of time-lock puzzles, and we use this below to complete the transformation.

**Construction.** Let $(\mathsf{DF.Setup}, \mathsf{DF.SampleInput}, \mathsf{DF.Eval})$ be a secure delay function from $\{0,1\}^n \to \{0,1\}^m$, then,

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$. Sample $\mathsf{pp} \leftarrow \mathsf{DF.Setup}(1^\lambda)$.
- $\mathsf{SampleInput}(\mathsf{pp}) \to x$. Run $x' \leftarrow \mathsf{DF.SampleInput}(\mathsf{pp})$, sample random coins $r \in \{0,1\}^m$ and set $x = (x', r)$.
- $\mathsf{Eval}(\mathsf{pp}, x) \to y$. Parse $x$ as $(x', r)$ and run $y' \leftarrow \mathsf{DF.Eval}(\mathsf{pp}, x')$. Output $y \leftarrow \langle y', r \rangle$.

*Claim* 1. If DF is a delay function that is $t(\lambda)$- sequential (Definition 1), then the construction above is a pseudorandom delay function with $t(\lambda) - \omega(\log \lambda)$ sequential pseudorandomness (Definition 3) where $\lambda$ is the security parameter.[10]

*Proof.* We now state the lemma statement that the queries made by the extraction process for the Goldreich-Levin [GL89] reduction can be made in parallel (the statement and proof were analyzed by [AMZ24], we simply state it in our notation).

**Lemma 1** ([GL89] adapted)**.** *Let $\mathcal{A}$ be an algorithm such that,*

$$\Pr_{r \in \{0,1\}^m, x \in \{0,1\}^n} \left[ \mathcal{A}(1^n, f(x), r) = \oplus_{i=1}^m x_i \cdot r_i \right] \geq 1/2 + \varepsilon,$$

*then there exists an algorithm $\mathcal{B}$, where,*

$$\Pr_{x \in \{0,1\}^n} \left[ \mathcal{B}(1^n, f(x)) = x', f(x') = f(x) \right] \geq 1/\varepsilon.$$

*Additionally, the depth of the adversary $\mathcal{B}$ is at most the depth of the adversary $\mathcal{A}$ plus $O(\log(1/\varepsilon))$.*

*Proof.* Similar to the proof of [AMZ24], the algorithm $\mathcal{B}$ samples $T = \mathsf{poly}(1/\varepsilon)$ many values and queries the algorithm $\mathcal{A}$ on $T$ locations in parallel. Finally, the algorithm $\mathcal{B}$ outputs a majority in time $\log T$ and outputs the majority guess for each independent bit of the input. Thus, the depth of $\mathcal{B}$ is depth of $\mathcal{A}$ plus the depth it takes to compute majority of $T$ many samples. □

Let $\mathcal{A}$ be an adversary that breaks the sequential randomness property in depth $\leq t(\lambda) - \omega(\log \lambda)$ with non-negligible $\varepsilon$. Thus, given $1^\lambda, \mathsf{pp}, \alpha, x, r_b$ it can guess bit $b$ with probability $1/2 + \varepsilon$ in the pseudorandom DF game where $\varepsilon = 1/\lambda^{O(1)}$. Then, we can run the algorithm $\mathcal{B}$ that runs in depth of $\mathcal{A}$ plus poly logarithmic in $1/\varepsilon$ breaks the delay function sequentiality in depth $\leq t(\lambda) - \omega(\log \lambda) + O(\log(1/\varepsilon)) \leq t(\lambda) - \omega(\log \lambda) + O(\log \lambda) \leq t(\lambda)$.

□

[10] Throughout the paper, we assume $t(\lambda) = \Omega(\lambda)$, allowing us to omit small terms from the depth bound of the adversary.

## 3.3 Coin Flipping Protocols

We closely follow the definition of distributed randomness beacons against sequentially-bounded adversaries in [BBCE25].

**Definition 5** (Coin Flipping Protocols)**.** A *coin flipping protocol* is a tuple of polynomial-time protocols $\mathsf{CoinFlip}_{\mathcal{P}} = (\mathsf{Setup}, \mathsf{Preprocess}, \langle\mathsf{Post}\rangle, \mathsf{Finalize})$ run by participants $\mathcal{P} = \{P_1, \ldots, P_n\}$ such that:

$\mathsf{Setup}(1^\lambda) \to \mathsf{CRS}$**:** Setup is a randomized algorithm that takes as input the security parameter and outputs a common reference string. We assume that Setup is run by a trusted party and CRS is made available to all participants.

$\mathsf{Preprocess}_i(\mathsf{CRS}) \to \mathsf{state}_i$**:** Preprocess is a randomized algorithm run locally by each participant $P_i$ to produce a local state for each user $\mathsf{state}_i$.

$\langle\mathsf{Post}\rangle\,(\mathsf{state}_1, \ldots \mathsf{state}_n; r_1, \ldots, r_n) \to \left(\left(\mathsf{tr}_1^{\mathsf{post}}, \mathsf{out}_1\right), \ldots, \left(\mathsf{tr}_n^{\mathsf{post}}, \mathsf{out}_n\right)\right)$ **:** Post is an interactive protocol between particpants in $\mathcal{P}$ where for $i \in [n]$, party $P_i$ takes as input the CRS, $\mathsf{state}_i$ and outputs string $\mathsf{out}_i$. Additionally, let the transcript of the protocol (which is the communication sent and received by the $i^{\text{th}}$ party $P_i$) be denoted by $\mathsf{tr}_i^{\mathsf{post}}$.

$\mathsf{Finalize}(\mathsf{state}_i, \mathsf{tr}_i^{\mathsf{post}}, \mathsf{out}_i) \to \Omega_i$ **:** Finalize is a (possibly randomized) *algorithm* run by each participant $P_i$ on input the state $\mathsf{state}_i$, their local transcript of the protocol $\mathsf{tr}_i^{\mathsf{post}}$ and their output $\mathsf{out}_i$, it produces $\Omega_i$, the output of the coin flipping protocol according to party $P_i$.

Let $\mathsf{CoinFlip}_{\mathcal{P}}(1^\lambda) \to (\Omega_1, \ldots, \Omega_n)$ denote the output of the algorithm where party $P_i$ outptus $\Omega_i$ for every $i \in [n]$. $\mathsf{CoinFlip}_{\mathcal{A}_C, \mathcal{P}}(1^\lambda) \to (\Omega_1, \ldots, \Omega_n)$ denote the protocol when parties $C \subseteq \{P_1, \ldots P_n\}$ and $\mathcal{A}_C$ denotes the algorithm run by each party in $C$. We now define the properties of our coin flipping protocol.

**Definition 6** (Agreement)**.** A coin flipping protocol satisfies *agreement* if for any subset of corrupt parties $C \subseteq \{P_1, \ldots, P_n\}$, let $\mathcal{A}_C$ be a PPT adversary that sends and receives messages on behalf of parties in $C$, then there exists a negligible function, such that,

$$\Pr[\exists P_i, P_j \notin C, \Omega_i \neq \Omega_j \mid \mathsf{CoinFlip}_{\mathcal{A}_C, \mathcal{P}}(1^\lambda) \to (\Omega_1, \ldots, \Omega_n)] \leq \mathsf{negl}(\lambda),$$

where the probability is over the random coins of the complete coin flipping protocol.

**Definition 7** ($t(\cdot)$-indistinguishability)**.** The $t(\cdot)$-indistinguishability game tasks a $t(\cdot)$-sequentially bounded adversary with distinguishing between a truly random string and the output of the given protocol. The adversary's computation is broken down into two phases. In the precomputation phase, the adversary declares all but one party $i$ as corrupted[11] and may run a polynomial-time algorithm $\mathcal{A}_0$ to produce an advice string $\alpha$. It passes this to $\mathcal{A}_1$ which is bounded to run in parallel time at most $O(t(\lambda))$. $\mathcal{A}_1$ engages in the Post protocol, controlling the corrupted parties. This game, denoted $\mathcal{G}^{\mathsf{indist}}$, is given in Figure 2.

A CoinFlip protocol satisfies *indistinguishability against a $t(\cdot)$-sequentially bounded adversary* if for all $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ (where $\mathcal{A}_0$ is PPT, $\mathcal{A}_1$ is constrainted to run in parallel time $O(t(\lambda))$),

$$\left|\Pr\left[\mathcal{G}^{\mathsf{indist}}_{\mathcal{A}, t, \mathsf{CoinFlip}}(1^\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

[11] This is strongest, we can define the more general notion where adversary outputs some corrupted set.

$\mathcal{G}^{\text{indist}}_{\mathcal{A},\mathcal{P},t,\mathsf{CoinFlip}}(1^\lambda)$

$P_{ist} \leftarrow \mathcal{A}_0(1^\lambda)$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\alpha_0, \{\mathsf{state}_i\}_{i\in\mathcal{P}\setminus\{P_{ist}\}}) \leftarrow \mathcal{A}_0(\mathsf{CRS})$,
$\mathsf{state}_{ist} \leftarrow \mathsf{Preprocess}_{ist}(\mathsf{CRS})$
$\left((\mathsf{tr}_1^{\text{post}}, \mathsf{out}_1), \ldots, (\mathsf{tr}_n^{\text{post}}, \mathsf{out}_n)\right) \leftarrow \langle \mathsf{Post}\rangle_{\mathcal{A}_1(\alpha_0),P_{ist}}(\mathsf{state}_1, \ldots, \mathsf{state}_n)$
$\Omega_{ist} \leftarrow \mathsf{Finalize}(\mathsf{state}_{ist}, \mathsf{tr}_{ist}^{\text{post}}, \mathsf{out}_{ist})$
$\Omega' \leftarrow \mathcal{A}_1(\alpha_0)$
$\texttt{return } \Omega_{ist} \stackrel{?}{=} \Omega'$

Figure 2: Security game for $t(\cdot)$-indistinguishability

*Remark* 1. We focus on the special case of coin flip protocols that output a single bit (following [Cle86] and [BBCE25].). In this case, negligible statistical distance coincides with computational indistinguishability.

*Remark* 2. Looking ahead, our constructions will have no setup, but we keep the more general definition for describing a larger class of protocols.

## 3.4 Byzantine Broadcast

We copy, with small modifications, the preliminaries on Byzantine broadcast from [BBCE25].

We rely on the well-known primitive of Byzantine Broadcast (BB) to remove the assumption of a public bulletin board and design a DRB secure in the dishonest majority setting. Specifically, in the BB problem there is an identified sender $s$, whose identity is known to all players, holding an input $v$. The goal is to design a protocol $\Pi$ that satisfies the following in the presence of $f$ byzantine failures (which could potentially include $s$) amongst the $n$ players. We refer to the players experiencing such failures as *corrupt* players. We refer to non corrupt players as *honest*.

1. **Termination.** Every honest player produces an output and halts after a finite number of rounds.
2. **Agreement.** All honest parties output the same value, provided that the corrupted parties run in parallel time $O(t(\lambda))$.
3. **Validity.** If $s$ is honest, then all honest parties output $v$.

**PKI model** Each party that is participating has a distinct public-private key pair $(\mathsf{pk}, \mathsf{sk})$. $\mathsf{pk}$ is known to all nodes at the start of the protocol.

## 3.5 Fine-grained Commitments with Preprocessing

**Definition 8** (Commitment against $t(\lambda)$-depth bounded adversaries)**.** A commitment secure against $t(\lambda)$-depth bounded adversaries is a tuple of p.p.t. algorithms $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Decom})$ where:

$\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$**:** takes as input the security parameter and outputs public parameters $\mathsf{pp}$.

$\mathsf{Com}(\mathsf{pp}, m) \to c, r$**:** is a randomized algorithm taking as input the public parameters and a message $m$. It outputs a commitment $c$ to the message, and the randomness $r$ used to compute $c$.[12]

[12]Slightly nonstandardly, we allow the commitment scheme to specify any polynomial-time samplable distribution over randomness $r$; $r$ may not be uniformly random.

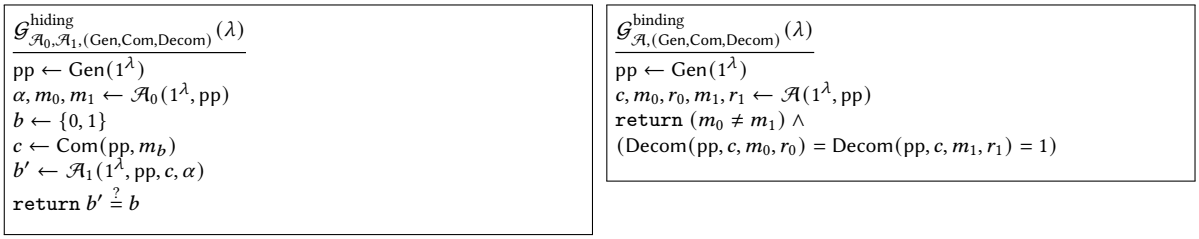



Figure 3: Commitment hiding and binding games

$\text{Decom}(\text{pp}, c, m, r) \to \{0,1\}$**:** takes as input the public parameters, a commitment, a message, and randomness, and outputs 1 if and only if the commitment opens to that message.

The commitment must satisfy $t(\cdot)$-*bounded hiding*, *binding* and *correctness*, defined below (Definitions 9 and 10). We emphasize that while hiding is guaranteed only against bounded depth adversaries, Com and Decom may require greater (though still polynomial) depth to compute.

**Definition 9** ($t(\cdot)$-bounded hiding)**.** A commitment scheme $(\text{Gen}, \text{Com}, \text{Decom})$ is $t(\cdot)$-bounded hiding if for all randomized algorithms $\mathcal{A}_0, \mathcal{A}_1$ running in total time $\text{poly}(\lambda)$, and $\mathcal{A}_1$ running in parallel time at most $O(t(\lambda))$,

$$\left|\Pr\left[\mathcal{G}^{\text{hiding}}_{\mathcal{A}_0,\mathcal{A}_1,(\text{Gen},\text{Com},\text{Decom})}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \text{negl}(\lambda).$$

**Definition 10** (Computational binding)**.** A commitment scheme $(\text{Gen}, \text{Com}, \text{Decom})$ is computationally binding if for all p.p.t. algorithms $\mathcal{A}$ (with arbitrary polynomial parallelism),

$$\Pr\left[\mathcal{G}^{\text{binding}}_{\mathcal{A},(\text{Gen},\text{Com},\text{Decom})}(\lambda) = 1\right] \leq \text{negl}(\lambda).$$

This is the standard definition of binding for commitments, and places no depth restriction on the adversary.

**Definition 11** (Correctness)**.** A commitment scheme $(\text{Gen}, \text{Com}, \text{Decom})$ is perfectly correct if for all messages $m$,

$$\Pr_{\text{pp}\leftarrow\text{Gen}(1^\lambda)}\left[\text{Decom}(\text{pp}, c, m, r) \to 1 : c, r \leftarrow \text{Com}(\text{pp}, m)\right] = 1.$$

**Definition 12** (Slow opening)**.** A commitment scheme $(\text{Gen}, \text{Com}, \text{Decom})$ has slow opening if there is some p.p.t. algorithm SlowOpen such that for all p.p.t. adversaries $\mathcal{A}$,

$$\Pr_{\text{pp}\leftarrow\text{Gen}(1^\lambda)}\left[\text{Decom}(\text{pp}, c, m, r) \neq 1 : c, m, r \leftarrow \mathcal{A}(\text{pp}), m, r \leftarrow \text{SlowOpen}(\text{pp}, c)\right] \leq \text{negl}(\lambda).$$

In other words, anyone can decommit using SlowOpen, without knowing $m$ or $r$. Observe that $t(\cdot)$-bounded hiding implies that SlowOpen is slow to evaluate.

We now show that a pseudorandom delay function naturally yields a commitment scheme that is hiding against similarly depth-bounded adversaries. Here, we define the scheme DFCom which is constructed using a delay function DF:

DFCom[DF = (Setup, SampleInput, Eval)]:

$\mathsf{Gen}(1^\lambda)$**:** outputs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.

$\mathsf{Com}(\mathsf{pp}, m)$**:** Samples $r \leftarrow \mathsf{SampleInput}(\mathsf{pp})$ and outputs $c = \mathsf{Eval}(\mathsf{pp}, r) \oplus m$ and $r$.

$\mathsf{Decom}(\mathsf{pp}, c, m, r)$**:** Outputs 1 if and only if $\mathsf{Eval}(\mathsf{pp}, r) \oplus c = m$.

**Theorem 1.** *If* DF *is a $t(\cdot)$-sequential pseudorandom delay function outputting $k$ bits,* DFCom[DF] *is a $t(\cdot)$-bounded hiding commitment scheme for $k$-bit messages.*

*Proof.* **Binding.** This commitment scheme is perfectly binding, with probability 1. Fix any pp and $c, r$ produced by the adversary. Decom outputs 1 only if $\mathsf{Eval}(\mathsf{pp}, r) \oplus c = m$. Since Eval is a deterministic function, this is true for only a single $m$.

**Hiding.** Suppose there exist adversaries $(\mathcal{A}_0, \mathcal{A}_1)$ winning $\mathcal{G}^{\text{hiding}}$ with non-negligible probability. We'll construct adversaries $(\mathcal{B}_0, \mathcal{B}_1)$ breaking $t(\cdot)$-sequential pseudorandomness of DF. $\mathcal{B}_0$ simulates $\mathcal{A}_0$ to produce $\alpha = (\alpha_{\mathcal{A}}, m_0, m_1)$. $\mathcal{B}_1$ receives from its challenger $r$, which is either uniform randomness or a delay function output. $\mathcal{B}_1$ draws a random bit $b$ and passes $\mathsf{pp}, c = r \oplus m_b$, and $\alpha_{\mathcal{A}}$ to $\mathcal{A}_1$. $\mathcal{B}_1$ outputs 1 if and only if $\mathcal{A}_1$ outputs $b$.

Observe that if $r$ was uniformly random, $r \oplus m_b$ statistically hides $m_b$ and $\mathcal{A}_1$ outputs $b$ with probability exactly $\frac{1}{2}$. If $r$ was a delay function output, this game is exactly the commitment hiding game and $\mathcal{A}_1$ outputs $b$ with non-negligible advantage. Therefore, $(\mathcal{B}_0, \mathcal{B}_1)$ succeed in distinguishing random $r$ from delay function $r$ with non-negligible advantage. $\mathcal{B}_0$ runs in polynomial time, and $\mathcal{B}_1$ runs in polynomial time and parallel time $t(\lambda) + O(1)$.

**Correctness.** For public parameters pp, given $m$ the commitment algorithm outputs $r$ and $c = \mathsf{Eval}(\mathsf{pp}, r) \oplus m$. Therefore, $\mathsf{Eval}(\mathsf{pp}, r) \oplus c = m$, and Decom will always output 1.

**Slow opening.** Let $\mathsf{SlowOpen}(\mathsf{pp}, c)$ parse $c$ to get $r, y$. It computes $m = \mathsf{Eval}(\mathsf{pp}, r)$ and outputs $(m, r)$. One can see by inspection that $\mathsf{Decom}(\mathsf{pp}, c, m, r) = 1$. □

## 3.6 Fine-grained Zero-Knowledge Proof of Knowledge with Preprocessing

Let $\mathcal{L} \in \mathsf{NP}$ be a language, and let $\mathcal{R}_{\mathcal{L}}$ denote the corresponding relation. We will consider interactive proofs of the following form:

**Definition 13.** Let $\mathcal{L} \in \mathsf{NP}$. A $t(\cdot)$-bounded interactive proof is a tuple of algorithms $\Pi = (\mathsf{Setup}, P, V)$ where:

$\mathsf{Setup}(1^\lambda) \to \mathsf{CRS}$**:** Setup takes as input the security parameter and outputs a common reference string CRS.

Prover: The prover $P = (P_0, P_1)$ consists of two algorithms. $P_0$ is an arbitrary p.p.t. algorithm which takes as input $(\mathsf{CRS}, x, w)$ and outputs an advice string $\alpha_P$. $P_1$ is p.p.t., running in parallel time $O(t(\lambda))$, and takes as input $(\mathsf{CRS}, x, w)$. $P_1$ interacts with the verifier; its goal is to convince the verifier that $x \in \mathcal{L}$.

Verifier: The verifier $V = (V_0, V_1)$ consists of two algorithms. $V_0$ is a p.p.t. algorithm of unbounded depth, which takes as input CRS and outputs an advice string $\alpha_V$. $V_1$ is p.p.t., running in parallel time $O(t(\lambda))$, and takes as input $(\mathsf{CRS}, x, \alpha_V)$. $V_1$ interacts with the prover and outputs 0 or 1.

$\mathcal{G}^{\mathrm{zk}}_{\mathcal{A}_0,\mathcal{A}_1,\Pi,S_0,S_1,(x,w)}(\lambda)$

$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}_0(\mathsf{CRS})$
$\beta \leftarrow S_0(\mathsf{CRS})$
If $b = 0$, $b' \leftarrow \mathrm{out}_{\mathcal{A}_1}\langle P(\mathsf{CRS}, x, w), \mathcal{A}_1(\mathsf{CRS}, x, \alpha)\rangle$
If $b = 1$, $b' \leftarrow \mathrm{out}_{\mathcal{A}_1}\langle S_1(\mathsf{CRS}, x, \beta), \mathcal{A}_1(\mathsf{CRS}, x, \alpha)\rangle$
`return` $b' \stackrel{?}{=} b$

Figure 4: Zero knowledge game

*Remark* 3. We define $V$ to allow a preprocessing algorithm $V_0$ for the sake of generality. However, the interactive proof we consider involves only $V_1$. Similarly, the setup phase sampling a CRS is allowed for generality, our interactive protocol will have no setup.

For a (possibly malicious) prover $P^*$, we let $\mathrm{out}_V\langle P^*, V\rangle(\mathsf{CRS}, x, w)$ denote $V's$ output at the end of the interaction between $P^*$ and $V$ on input $(\mathsf{CRS}, x, w)$. More generally, for two interacting parties $A$ and $B$, we let $\mathrm{out}_B\langle A, B\rangle(\cdot)$ denote the output of $B$ at the end of the interaction on input $\cdot$. If $A$ and $B$ take different inputs $a$ and $b$, we write $\mathrm{out}_B\langle A(a), B(b)\rangle$.

We will require several properties of the time-bounded interactive proof we use.

**Definition 14** (Soundness)**.** An interactive proof $\Pi$ for a language $\mathcal{L} \in \mathsf{NP}$ satisfies *soundness* if for every $x \notin \mathcal{L}$, every $w$, and every cheating (unbounded-time) prover $P^*$,

$$\Pr_{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)}[1 \leftarrow \mathrm{out}_V\langle P^*, V\rangle(\mathsf{CRS}, x, w)] \leq \mathsf{negl}(\lambda).$$

**Definition 15** (Perfect completeness)**.** An interactive proof $\Pi$ for a language $\mathcal{L} \in \mathsf{NP}$ satisfies *perfect completeness* if for every $(x, w) \in \mathcal{R}_{\mathcal{L}}$,

$$\Pr_{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)}[0 \leftarrow \mathrm{out}_V\langle P, V\rangle(\mathsf{CRS}, x, w)] = 0.$$

**Definition 16** (Zero knowledge to bounded-depth verifiers)**.** An interactive proof $\Pi$ for $\mathcal{L}$ satisfies *zero-knowledge to $t(\cdot)$-depth-bounded verifiers* if there exists a simulator $\mathcal{S} = (S_0, S_1)$ where:

- $S_0$ is a p.p.t. algorithm of unbounded depth, and $S_1$ is a p.p.t. $o(t(\lambda))$-depth-bounded algorithm.
- For any $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ where $\mathcal{A}_0$ is a p.p.t. algorithm of unbounded depth, and $\mathcal{A}_1$ is a p.p.t. $t(\cdot)$-depth-bounded algorithm, for any $(x, w) \in \mathcal{R}_{\mathcal{L}}$,

$$\left|\Pr\left[\mathcal{G}^{\mathrm{zk}}_{\mathcal{A}_0,\mathcal{A}_1,\Pi,S_0,S_1,(x,w)}(\lambda) = 1\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

In other words, there is a simulator consisting of an unbounded-depth preprocessing algorithm taking as input the CRS, and a bounded-depth interactive algorithm that receives $x$ but not $w$. No bounded-depth adversarial verifier can distinguish between the case where it is interacting with the simulator, versus the honest prover that depends on both $x$ and $w$. It will be useful for our applications that the interactive simulator is depth-bounded.

We require a fairly strong notion of bounded-depth extraction. At a high level, there is an extractor that can extract a valid witness from any prover that convinces the verifier with non-negligible probability. This extractor is $o(t(\lambda))$-depth-bounded, where each query to the prover counts as 1 operation regardless of the prover's depth.

**Definition 17** (Bounded-depth extraction)**.** A protocol $\Pi$ is knowledge sound if there exists a p.p.t. extractor $\mathcal{E}$ such that for any (cheating) prover $P^*$, for any input $x$,

$$\Pr_{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)}\left[\mathsf{out}_V\langle P^*(\mathsf{CRS},x),V(\mathsf{CRS},x)\rangle=1\right]=\alpha,$$

then,

$$\Pr_{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)}\left[(x,w)\in\mathcal{R}_{\mathcal{L}}:\ w\leftarrow\mathcal{E}^{P^*}(\mathsf{CRS},x)\right]\geq\alpha-\mathsf{negl}(\lambda).$$

Furthermore, $\mathcal{E}$ runs in parallel time at most $O(t(\lambda))$.

Here, $\mathcal{E}^{P^*}$ means that $\mathcal{E}$ has black-box access to the prover, and can rewind the prover and query it for its next message. Additionally, we assume that the prover runs in *deterministic* polynomial time in the length of their input and takes in an advice string of $\mathsf{poly}(\lambda)$ size. We can derandomize any cheating prover by fixing its random coins as advice to the choice that maximizes the prover's probability of convincing the verifier.

In the following definition, we let $(\mathsf{tr},\mathsf{out}_B)\langle A,B\rangle$ denote the transcript tr and output of $B$ $\mathsf{out}_B$ of an interaction between $A$ and $B$. The transcript includes all messages sent over the network. In particular, it does *not* include advice strings, e.g. those passed between verifier algorithms.

$\Pi$ has robust public verification if there is a function $V_{\mathsf{pub}}$ computable from only the CRS, input $x$, and transcript of the protocol such that (1) $V_{\mathsf{pub}}$ agrees with $V$ with overwhelming probability, and (2) an honest prover can always induce a transcript causing $V_{\mathsf{pub}}$ to accept, even when faced with a malicious verifier. More formally:

**Definition 18** (Robust public verification)**.** $\Pi$ has robust public verification if there exists an algorithm $V_{\mathsf{pub}}$ such that for any cheating prover $P^*$, for any $x$,

$$\Pr_{\mathsf{CRS}\leftarrow\mathsf{Setup}(1^\lambda)}\left[V_{\mathsf{pub}}(\mathsf{CRS},x,T)\neq b:\ (T,b)\leftarrow(\mathsf{tr},\mathsf{out}_V)\langle P^*,V\rangle(\mathsf{CRS},x)\right]\leq\mathsf{negl}(\lambda).$$

Furthermore, for any malicious *verifier* $V^*$, for any $(x,w)\in\mathcal{R}_{\mathcal{L}}$,

$$\Pr\left[V_{\mathsf{pub}}(\mathsf{CRS},x,T)=1:\ (T,b)\leftarrow(\mathsf{tr},\mathsf{out}_V)\langle P(\mathsf{CRS},x,w),V^*(\mathsf{CRS},x)\rangle\right]=1.$$

**Definition 19.** We will consider the language $\mathcal{L}_{\mathsf{com}}$:

$$\mathcal{L}_{\mathsf{com}}=\{(\mathsf{pp}^*,c)\ :\ \exists m\text{ s.t. }\mathsf{DFCom.Decom}(\mathsf{pp}^*,c,m,\bot)=1\}.$$

Let HamCycle be the language of directed graphs with Hamiltonian cycles.

**Nice reduction.** We let $\sigma$ be a reduction from $\mathcal{L}_{\mathsf{com}}$ to HamCycle with the following properties:

- $\sigma$ takes as input $(\mathsf{pp}^*,c)$ and optionally a message $m$. Given only $(\mathsf{pp}^*,c)$, $\sigma$ outputs a graph $\mathcal{G}$ that has a Hamiltonian cycle if and only if there exists $m$ such that $\mathsf{DFCom.Decom}(\mathsf{pp}^*,c,m,\bot)=1$. Given $(\mathsf{pp}^*,c)$ and $m$ such that $\mathsf{DFCom.Decom}(\mathsf{pp}^*,c,m,\bot)=1$, $\sigma$ outputs $\mathcal{G}$ and a Hamiltonian cycle $H$.

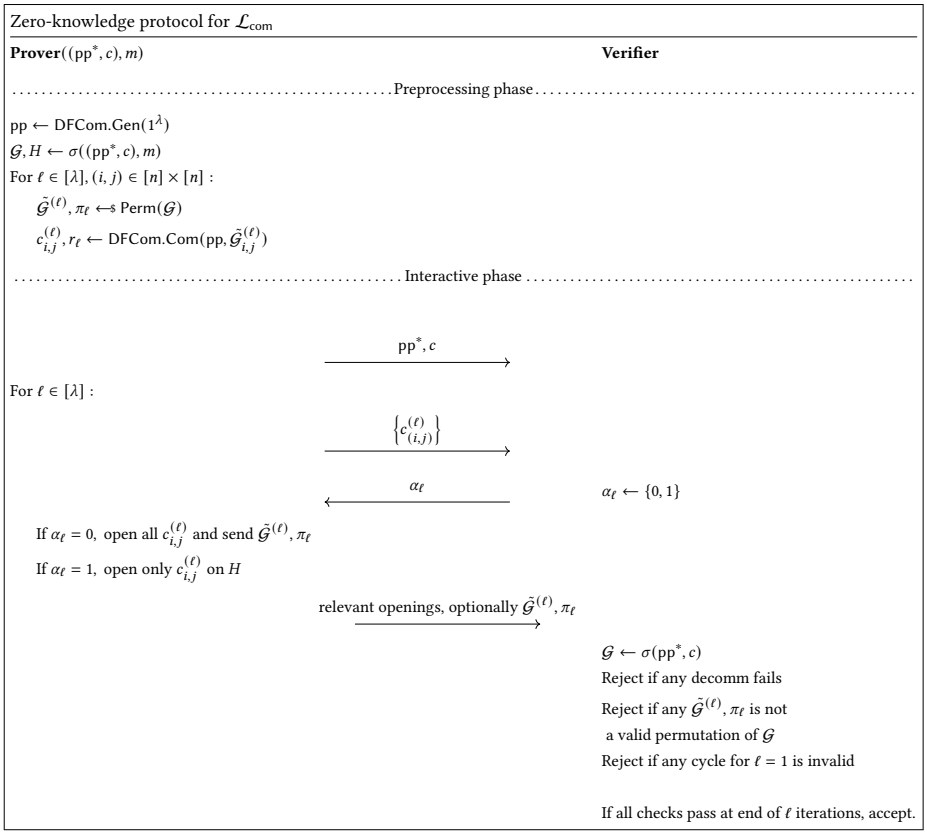



Figure 5: Here, we use DFCom with delay parameter $t(\lambda)$. This scheme is based on a folklore proof of Hamiltonicity of Blum, modified to use fine-grained commitments and enable low-depth verification.

- Given only $(\mathsf{pp}^*, c)$, one can compute the corresponding $\mathcal{G} \leftarrow \sigma((\mathsf{pp}^*, c), \cdot)$ in constant parallel time. Given both $(\mathsf{pp}^*, c)$ and $m$, one can compute the graph and Hamiltonian cycle $\mathcal{G}, H \leftarrow \sigma((\mathsf{pp}^*, c), m)$ in constant parallel time. The algorithm is permitted to use (arbitrary depth) polynomial-time preprocessing given $1^\lambda$, before seeing $(\mathsf{pp}^*, c)$.
- Given any Hamiltonian cycle $H$ in $\mathcal{G}$, one can extract the corresponding $m$ in low depth.

We observe that the standard reduction has these properties and recall it briefly here. Consider a circuit $C$ computing $\mathsf{DFCom.Decom}(\cdot, \cdot, \cdot, \bot)$ consisting of AND/OR gates of fan-in 2, and NOT gates. First, observe that $C$ can be transformed into an equivalent 3SAT formula with one variable for each input bit and each gate. This 3SAT formula can be precomputed in the preprocessing phase, given only $1^\lambda$.

The reduction from 3SAT to HamCycle computes a "row gadget" for each variable and a vertex for each clause (see Figure 6). A clause gadget is connected to each of its relevant variables' row gadgets, as shown in the figure. The rows gadgets are then stacked, and directed edges are added between the ends of each rows. Any Hamiltonian cycle of this graph must traverse every row exactly once. The direction in which it traverses each row determines the assignment of that variable in the satisfying assignment. We

refer the reader to lecture notes for further details.[13]

We now argue that this reduction has the desired efficiency properties. To compute $\sigma$, one first precomputes the described graph for the circuit $C$. This graph contains one row gadget for each input bit. Once the inputs $(\mathsf{pp}^*, c)$ and optionally $m$ are specified, one can in low depth amend the corresponding row gadgets to fix those variables. That is, if the first bit of $\mathsf{pp}^*$ is 1, we delete the "false" edges of that row gadget, forcing that variable to "true". This can be done in parallel for each input bit.

To extract an $m$ from a Hamiltonian cycle $H$, one computes each bit of $m$ in parallel by examining the corresponding row gadget. That is, arbitrarily select some canonical node in the middle of the gadget. One can check whether its right-pointing or left-pointing outgoing edge is in the cycle. If right-pointing, that variable is "false"; if left-pointing, it is "true". This check can be done in constant depth.

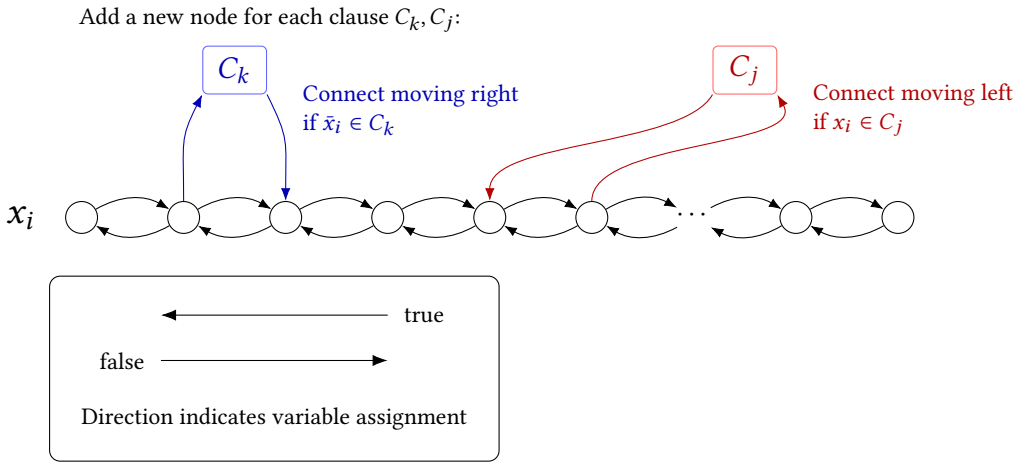



Figure 6: Variable gadget for the reduction from 3SAT to HamCycle.

**Theorem 2.** *The protocol given in Figure 5 satisfies soundness, perfect completeness, zero knowledge to $t(\lambda)$-depth-bounded verifiers, bounded-depth extraction, and robust public verification. The prover and verifier are bounded depth when needed.*

We separate the proof of this theorem into lemmas below.

**Lemma 2.** *The protocol given in Figure 5 satisfies soundness and perfect completeness.*

*Proof.* **Soundness.** Suppose that $C \notin \mathcal{L}$. Then no permutation of $\mathcal{G} = \sigma(C)$ has a Hamiltonian cycle. Consider each commitment $c^{(i)}$ sent by the prover. Perfect binding implies that there is a unique value that this commitment can be decommitted to. Therefore, this value is either a permutation of $\mathcal{G}$, in which case it has no Hamiltonian cycle, or it is not a valid permutation of $\mathcal{G}$. With probability $1/2$, the prover cannot open to a value satisfying $\alpha_i$. The probability of failure is independent for each $\alpha_i$; therefore, the probability that the prover satisfies the verifier in all openings is $2^{-\lambda}$.

**Perfect completeness.** We defer this proof to Lemma 4, where we show a stronger property that the prover convinces the verifier even if the challenge $\alpha$ is maliciously chosen. □

[13] `https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/sat.pdf`

**Lemma 3.** *The protocol given in Figure 5 satisfies zero-knowledge to $t(\lambda)$-depth-bounded verifiers.*

*Proof.* We recall the standard simulator $\mathcal{S}$ of Blum's protocol, for a single iteration of the interactive phase. To simulate a single iteration, $\mathcal{S}$ repeats the following $\lambda$ many times:

1. Guess $c' \leftarrow \{0,1\}$
2. If $c' = 0$, commit to a random permutation of $\mathcal{G}$. If $c' = 1$, commit to a random cycle.
3. Send the commitments to the verifier, who responds with $c \in \{0,1\}$.
4. If $c = c'$, open the corresponding commitment, and terminate and output this transcript. Else, continue.

We refer the reader to [Gol01] for a full proof.

We modify $\mathcal{S}$ for our setting, constructing a simulator consisting of a pair of algorithms $(S_0, S_1)$. Recall that $S_0$ is a preprocessing algorithm that is not depth-bounded; $S_1$ has depth at most $t(\lambda)$.

$S_0$ samples $\mathsf{pp} \leftarrow \mathsf{DFCom.Gen}(1^\lambda)$. $S_0$ then samples enough random inputs $r_1, r_2, \ldots$ to generate all commitments used in the protocol. It precomputes $y_i = \mathsf{Eval}(\mathsf{pp}, r_i)$ for all $i$. For each iteration $\ell$, it also samples random permutations $\pi_1^{(\ell)}, \ldots, \pi_\lambda^{(\ell)}$. It outputs an advice string $\beta$ consisting of pp and $(r_i, y_i)$ for all $i$.

$S_1$ receives the input $(C, \mathcal{G}, \mathsf{tab}(\sigma(C)))$ and for each iteration repeats the following $\lambda$ many times, in parallel:

1. Guess $c' \leftarrow \{0,1\}$
2. If $c' = 0$, commit to a random permutation of $\mathcal{G}$. If $c' = 1$, commit to a random cycle. Each commitment to the bit $b$ of the adjacency matrix is equal to $b \oplus y_i$ for some precomputed $y_i$.
3. Send the commitments to the verifier, who responds with $c \in \{0,1\}$.
4. For the "first" (according to some canonical ordering) iteration where $c' = c$, compute the corresponding commitment openings and terminate and output this transcript.

**$S_1$ is bounded-depth.** The random permutations of $\mathcal{G}$ can be computed in parallel. The commitments to each bit of these adjacency matrices can also be computed in parallel, and committing to each bit simply involves XORing a precomputed $y_i$. Opening the commitments can also be done in parallel, and involves sending the corresponding $y_i$'s and $r_i$'s. Doing this in sequence for all $\lambda$ iterations takes $\lambda$ sequential steps.

**$(S_0, S_1)$ is a zero-knowledge simulator.** We recall the proof that $\mathcal{S}$ is a zero-knowledge simulator, adapting it to our time-bounded setting. First, note that with overwhelming probability $S_1$ succeeds at guessing $c' = c$ in at least one trial. Now, observe that if $c = 0$, $S_1$'s messages are distributed identically to the honest prover's: It commits to a random permutation of the actual adjacency matrix of $\mathcal{G}$. If $c = 1$, $S_1$ committed to a random cycle and opened commitments only to edges on this cycle. Suppose a bounded-depth $\mathcal{A}_1$ can distinguish between $P$ and $S_1$ here. Then, $\mathcal{A}_1$ can distinguish between whether the unopened commitments are to 0, or to edges from $\mathcal{G}$. $\mathcal{A}_1$ would therefore break hiding of the time-bounded commitments. □

**Lemma 4.** *The protocol given in Figure 5 satisfies robust public verification with function $V_{\mathsf{pub}}$.*

*Proof.* **Specifying the public verifier.** $V_{\mathsf{pub}}$ takes as input pp, $C$, $\{c_{i,j}^{(1)}\}, \ldots, \{c_{i,j}^{(\lambda)}\}$, $\alpha$, and the permutations and openings sent by the verifier. $V_{\mathsf{pub}}$ computes in parallel $\mathcal{G} = \sigma(C)$.

For each challenge $\alpha_\ell = 0$, $V_{\mathsf{pub}}$ checks in parallel that the corresponding $\pi_\ell$ is a valid permutation, and $\tilde{\mathcal{G}}^{(\ell)}$ is the output of the permutation applied to $\mathcal{G}$. It then checks that the opened commitments are valid and match $\tilde{\mathcal{G}}^{(\ell)}$.

For each challenge $\alpha_\ell = 1$, $V_{\mathsf{pub}}$ checks that the decommitments are valid. It then checks that the decommitted values give a Hamiltonian cycle as follows, using high parallelism. We can represent this function using a circuit, which computes $n^2$ boolean flags as an intermediate step. Flag $(i, j)$ is set to 1 if and only if entry $(i, j)$ of the adjacency matrix is decommitted, and no other row has entry $j$ decommitted. That is, flag $(i, j)$ is computed as the AND of (entry $(i, j)$ is decommitted) and for all $j' \neq j$, (entry $(i, j)$ is not decommitted). One can see that these flags can be computed in parallel, in constant depth. The AND of these flags is 1 if and only if each column has exactly one decommitment. $V_{\mathsf{pub}}$ repeats this process with $n^2$ additional flags, where each now checks that each *row* has exactly one decommitment. The output is the AND of all of these flags.

**Public verifier agrees with the verifier.** By construction, the public verifier is exactly the same as the verifier.

**Malicious verifier cannot cause rejection.** A malicious verifier has freedom only in specifying its challenge $\alpha$. For every possible value of $\alpha_\ell$ for each $\ell$, an honest prover can open its committed adjacency matrix correctly. Therefore, public verification will always see a valid permutation when $\alpha = 0$, and a valid Hamiltonian cycle when $\alpha = 1$. □

## 3.7 Broadcast with PKI from Delay Functions

**Definition 20** (One-time signatures with forgery prevention against $t(\lambda)$-depth bounded adversaries)**.** A one time signature scheme with forgery prevention against $t(\lambda)$-depth bounded adversaries is a tuple of p.p.t. algorithms (Setup, Sign, Verify) where:

$\mathsf{Setup}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$**:** takes as input the security parameter and outputs a verification key and a signing key.

$\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$**:** is a randomized algorithm taking as input the secret key and a message $m$ and outputs a signature $\sigma$.

$\mathsf{Verify}(\mathsf{vk}, \sigma, m) \to \{0, 1\}$**:** takes as input the verification key, signature and message and outputs $\{0, 1\}$ indicating if the signature is valid or not.

The one time signature must satisfy correctness and one-time security defined below (Definitions 9 and 10). Additionally, the verification algorithm should be bounded-depth efficient, i.e. it should run in parallel time $o(t(\lambda))$.

**Definition 21** (Correctness)**.** A one-time signature scheme (Setup, Sign, Verify) is said to be correct if for all (vk, sk) in support of Setup, all messages $m$,

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, \sigma, m) = 1 : \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)\right] = 1,$$

where the randomness is over the random coins used by the Sign algorithm.

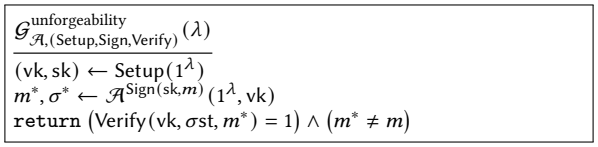



Figure 7: One-time existential unforgeability under chosen-message attack (EUF-OT-CMA) game for digital signatures.

**Definition 22** ($t(\cdot)$-bounded One-time existential unforgeability)**.** A signature scheme (Setup, Sign, Verify) is *one-time existentially unforgeable under chosen-message attack* if for all p.p.t. adversaries $\mathcal{A}$ running in parallel time $O(t(\lambda))$ and making at most one signing query,

$$\Pr\left[\mathcal{G}^{\text{unforgeability}}_{\mathcal{A},(\mathsf{Setup},\mathsf{Sign},\mathsf{Verify})}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Our construction follows directly from the construction of [Lam79], where instead of a one-way function, we use a delay function and argue security against $t(\cdot)$-bounded adversaries.

OTSign[DF = (Setup, SampleInput, Eval)]: Let the message space of the scheme be $m \in \{0,1\}^\ell$.

Setup($1^\lambda$)**:** outputs pp $\leftarrow$ Setup($1^\lambda$). Sample $\{x_{i,b}\}_{i\in[\ell],\, b\in\{0,1\}}$ with $x_{i,b} \leftarrow$ SampleInput(pp), and set images $y_{i,b} \leftarrow$ Eval(pp, $x_{i,b}$) and $\pi_{i,b}$ be a bounded depth proof of the the computation (computed as a tableau).

Output

$$\mathsf{vk} \leftarrow \left(\left(x_{i,0}, x_{i,1}\right)_{i\in[\ell]}, \mathsf{pp}\right), \qquad \mathsf{sk} \leftarrow \left(y_{i,0}, y_{i,1}, \pi_{i,0}, \pi_{i,1}\right)_{i\in[\ell]}.$$

Sign(sk, $m \in \{0,1\}^\ell$)**:** $\sigma_i \leftarrow \left(y_{i,m_i}, \pi_{i,m_i}\right)$ for each $i \in [\ell]$, $\sigma = (\sigma_1, \ldots, \sigma_\ell)$.

Verify(vk, $\sigma, m$)**:** Parse $\sigma$, vk appropriately and check if for all $i \in [\ell]$, using $\pi_{i,m_i}$ to verify in bounded depth that Eval(pp, $x_{i,m_i}$) = $y_{i,m_i}$. Observe that this bounded depth proof can be checked in constant parallel time via the tableau.

*Claim* 2. Let DF be a $t(\lambda)$-sequential delay function, then the construction above is correct and $t(\lambda)$-bounded one time existentially unforgeable.

*Sketch.* Correctness of the algorithm follows from the fact that $\forall i \in [\ell], b \in \{0,1\}$, Eval(pp, $x_{i,b}$) = $y_{i,b}$.

Unforgeability follows from the fact that the adversary must output $m^* \neq m$, thus there exists some position $i \in [\ell]$ where $m^*_i \neq m_i$. Thus, to reduce to the sequentiality of the delay function, the reduction chooses an index $i \in [\ell], b \in \{0,1\}$ randomly and programs $x_{i,b}$ as the challenge input. With probability $1/2\ell$, the adversary outputs $m$,$m^*$, such that, $m^*_i = b$ and $m_i = 1 - b$. □

*Claim* 3 ([DS83] Adapted). Let OTSign be a correct and $t(\lambda)$-bounded one-time existential unforgeable scheme. Additionally, let the algorithms Sign, Verify run in bounded depth parallel time $O(1)$, then there exists an authenticated protocol for $o(t)$ users that solves broadcast in the PKI model against any adversary controlling a dishonest majority where validity holds against time $t(\lambda)$-bounded adversaries and termination happens in $o(t)$ time.

*Sketch.* The proof follows exactly as [DS83], where,

1. **Termination.** Every honest player produces an output and halts after a finite number of rounds $r$. The quantity $r$ is upper bounded by the total number of corruptions allowed (and hence upper bounded by the total number of users participating in the protocol). Since the signing and verification is efficient take $O(1)$ depth, termination runs in parallel time $O(r)$. Thus, as long as $r = o(t)$, our algorithm terminates in bounded parallel time $o(t)$. Note that the protocol uses distinct instances of a one-time signature scheme in each round.

2. **Validity.** The (honest) leader will only sign one value, and this value will be sent to all honest parties in the first round. Due to the unforgeability against time $t$-adversaries, no other signature chain can exist. The same proof of [DS83] will work in our bounded depth setting.

3. **Agreement.** The proof that all honest parties output the same value is idential to the proof in [DS83].

□

**Multi-party coin flipping protocol**

Our protocol is setup-less.

Slow preprocessing phase, no time bound:

$\mathsf{Preprocess}_i(1^\lambda) \to \mathsf{state}_i$

1. All parties generate public parameters for the pseudorandom delay function, the commitment scheme, the interactive proof protocol and all preprocessing necessary for their commitments and proofs.
2. Each party $P_i$ samples a bit $v_i$ and generates $c_i \leftarrow \mathsf{DFCom}(\mathsf{pp}_i, v_i)$.
3. State $\mathsf{state}_i$ contains all the public parameters, the pre-processed information and commitment $c_i$.

Interactive phase, time bounded:

$\langle \mathsf{Post} \rangle(\mathsf{state}_1, \ldots, \mathsf{state}_n; r_1, \ldots, r_n)$.

3. Each party $P_i$ broadcasts $\mathsf{pp}_i, c_i$.
4. We run $n$ rounds sequentially, For $j \in [n]$, Round $j$ proceeds as follows.
   - For all $i \neq j$, $P_i$ proves knowledge of its committed value to $P_j$ in fine-grained zero-knowledge, broadcasting all messages, that $(\mathsf{pp}_i, c_i) \in \mathcal{L}_{\mathsf{com}}$.

Slow postprocessing phase, no time bound:

$\mathsf{Finalize}(\mathsf{state}_i, \mathsf{tr}_i^{\mathsf{Post}}, \mathsf{out}_i) \to \Omega_i$.

5. Party $P_i$ initializes flags $\phi_1^{(i)} = \ldots = \phi_n^{(i)} = 1$ and checks all proofs on behalf of all parties using $V_{\mathsf{pub}}$. For each $j \in [n]$, if $P_j$ sent a proof that rejects (to any party) or fails to send a verifier message (in any round): set flag $\phi_j^{(i)}$ to 0.
6. Party $P_i$ opens all commitments to obtain $v'_1, \ldots, v'_n$.
7. Party $P_i$ outputs $\Omega_i = \bigoplus_{j=1}^n \phi_j^{(i)} \cdot v'_j$.

Figure 8: Multi-party coin-flipping protocol. Uses a fine-grained commitment DFCom, and fine-grained zero knowledge proof $\Pi$.

# 4 Simple Multiparty Coin Flipping

Our protocol $\mathsf{CoinFlip}_{\mathcal{P}}$ is listed in Figure 8 and must satisfy agreement and $t(\cdot)$-indistinguishability properties. We show this below.

Our proof of $t(\cdot)$-indistinguishability relies on the fact that the zero-knowledge proofs *can be* verified in low depth, using high parallelism. However, we emphasize that in the actual protocol, honest parties verify the proofs in the slow postprocessing phase, in as little parallelism as they like.

*Claim* 4. Assuming the broadcast channel satisfies agreement, the ZK protocol follows robust verification, and the commitment scheme satisfies binding, our protocol satisfies agreement.

*Proof.* In the agreement experiment where an adversary $\mathcal{A}_C$ participates in the protocol for parties $C \subseteq \mathcal{P}$, correctness is broken if two honest parties compute different outputs.

Let there be two honest parties $P_\iota, P_{\iota^*} \in \mathcal{P}$ but $P_\iota, P_{\iota^*} \notin C$. We assume a broadcast channel wlog (see Lemma 3) and all communication happens over the broadcast channel. Since the broadcast channel satisfies agreement with probability 1, we can assume that all messages broadcasted are in agreement amongst $P_\iota, P_{\iota^*}$. We first show that both $P_\iota, P_{\iota^*}$ compute the same $\phi_j = \phi_j^{(\iota)} = \phi_j^{(\iota^*)}$ for every $j \in [n]$. Since each party $P_\iota$ verifies all the proofs sent by the party $P_j$ and the protocol satisfies robust public coin verification. Each party can compute the output of the interactive protocol given the CRS, the input statement and the transcript. Since the inputs to the function are the same, the output is the same. Finally, if $\phi_j$ is 1, we open the commitment, from the perfect binding of the commitment, each $c_j$ is opened to the same value $v'_j$ by

parties $P_\iota, P_{\iota^*}$, hence we have consistency. □

**Theorem 3.** *Let the following hold:*

- DFCom *is a fine-grained commitment that is computationally binding, correct, $t(\cdot)$-bounded hiding, and with slow opening, and*
- $\Pi$ *is a fine-grained zero-knowledge proof of knowledge, satisfying soundness, zero knowledge to $t(\cdot)$-bounded-depth verifiers, $t(\cdot)$-bounded-depth extraction, and robust public verification*[14].

*Then our n-party coin-flipping protocol in Figure 8 satisfies $t(\cdot)$-indistinguishability.*

*Proof.* Suppose there is an adversary, $\mathcal{A}$, who corrupts all parties but $P_{i^*}$ and yet can bias $P_{i^*}$'s output to 0 (other case is symmetrical). We can use $\mathcal{A}$ to break the security of our underlying primitives.

- $\mathsf{Hyb}_0$: Run real protocol where we set $v_{i^*} = 0$, and party $P_{i^*}$ outputs $\Omega_{i^*} = \sum_{j\in[n]} \phi_j^{(i^*)} \cdot v'_j$. We sketch the full protocol below for completeness.
  - $(\alpha_0, \{\mathsf{state}_i\}_{i\in\mathcal{P}\setminus\{P_{i^*}\}}) \leftarrow \mathcal{A}_0(1^\lambda)$,
  - $\mathsf{state}_{i^*} \leftarrow \mathsf{Preprocess}_{i^*}(1^\lambda)$
    1. Generate public parameters and preprocessing necessary for the commitments and proofs.
    2. Party $P_{i^*}$ sets $v_{i^*} = 0$ and generates $c_{i^*} \leftarrow \mathsf{DFCom}(\mathsf{pp}_{i^*}, v_{i^*})$.
    3. State $\mathsf{state}_i$ contains all the preprocessed information and commitment $c_i$.
  - $\left((\mathsf{tr}_1^{\mathsf{post}}, \mathsf{out}_1), \ldots, (\mathsf{tr}_n^{\mathsf{post}}, \mathsf{out}_n)\right) \leftarrow \langle \mathsf{Post}\rangle_{\mathcal{A}_1(\alpha), P_{i^*}}(\mathsf{state}_1, \ldots, \mathsf{state}_n)$
    3. Each party $P_i$ broadcasts $\mathsf{pp}_i, c_i$.
    4. For all $i \neq j$, $P_i$ proves to $P_j$ in fine-grained zero-knowledge (done sequentially), broadcasting all messages, that $(\mathsf{pp}_i, c_i) \in \mathcal{L}_{\mathsf{com}}$.
  - $\Omega_{i^*} \leftarrow \mathsf{Finalize}(\mathsf{state}_{i^*}, \mathsf{tr}_{i^*}^{\mathsf{post}}, \mathsf{out}_{i^*})$
    5. Party $P_{i^*}$ initializes flags $\phi_1^{(i^*)} = \ldots = \phi_n^{(i^*)} = 1$ and checks all proofs on behalf of all parties using $V_{\mathsf{pub}}$. For each $j \in [n]$, if $P_j$ sent a proof that rejects (to any party) or fails to send a verifier message (in any round): set flag $\phi_j^{(i^*)}$ to 0.
    6. Party $P_{i^*}$ opens all commitments to obtain $v'_1, \ldots, v'_n$.
    7. Party $P_{i^*}$ outputs $\Omega_{i^*} = \bigoplus_{j\in[n]} \phi_j^{(i^*)} \cdot v'_j$.
  - $\Omega' \leftarrow \mathcal{A}_1(\alpha_0)$
    `return` $\Omega_{i^*} \stackrel{?}{=} \Omega'$
- $\mathsf{Hyb}_a$: Same as $\mathsf{Hyb}_0$ but if $\phi_{i^*}^{(i^*)} = 0$ when computing Finalize, the experiment quits and returns 1. That is, if the honest party is unable to output valid proofs when interacting with the adversary, the adversary wins.
- $\mathsf{Hyb}_b$: Same as $\mathsf{Hyb}_a$, but when running Finalize, instead of opening all commitments $c_j$ brute force to compute $v'_j$, we rewind and use knowledge extractor. Additionally, we run the verification process for the proof system in low depth as stated in Section 3.6.

[14] This implies completeness.

- $\mathsf{Hyb}_c$: Same as $\mathsf{Hyb}_b$, except proofs for $(\mathsf{pp}_{i^*}, c_{i^*}) \in \mathcal{L}_{\mathsf{com}}$ are simulated using the ZK simulator.
- $\mathsf{Hyb}_x$: Same as $\mathsf{Hyb}_y$, except proofs for $(\mathsf{pp}_{i^*}, c_{i^*}) \in \mathcal{L}_{\mathsf{com}}$ are simulated using the ZK simulator.
- $\mathsf{Hyb}_y$: Same as $\mathsf{Hyb}_z$, but when running Finalize, instead of opening all commitments $v_j$ brute force, we rewind and use knowledge extractor. Additionally, we run the verification process for the proof system in low depth as stated in Section 3.6.
- $\mathsf{Hyb}_z$: Same as $\mathsf{Hyb}_1$ but if $\phi_{i^*}^{(i^*)} = 0$ when computing Finalize, the experiment quits and returns 1.
- $\mathsf{Hyb}_1$: Run real protocol with $v_{i^*} = 1$, and party $P_{i^*}$ outputs $\Omega_{i^*} = \sum_{j\in[n]} \phi_j^{(i^*)} \cdot v'_j$.

We write $\mathsf{Hyb}(\mathcal{A})$ to denote the distribution of the output of Hyb when run with adversary $\mathcal{A}$. In this proof, we consider security against *non-uniform* adversaries (indexed by $\lambda$) that run in *deterministic* polynomial time in the length of their input and takes in an advice string of $\mathsf{poly}(\lambda)$ size.[15] We analyze adjacent pair of hybrid distributions.

*Claim* 5. From robust verification of our ZK protocol, it follows that $\mathsf{Hyb}_0$ and $\mathsf{Hyb}_a$ are statistically close. Similarly, $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_z$ are statistically close.

*Proof.* We'll show the claim for $\mathsf{Hyb}_0$ and $\mathsf{Hyb}_a$; an analogous argument applies to $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_z$.

Recall that robust public verification states that for transcripts generated by an honest prover interacting with a malicious verifier on $(\mathsf{pp}, c) \in \mathcal{L}_{\mathsf{com}}$, $V_{\mathsf{pub}}$ accepts with probability 1. $\phi_{i^*}^{(i^*)}$ is derived using $V_{\mathsf{pub}}$ on the transcripts of $P_{i^*}$'s proofs to all malicious verifiers on input $(\mathsf{pp}_{i^*}, c_{i^*}) \in \mathcal{L}_{\mathsf{com}}$. $V_{\mathsf{pub}}$ outputs 1 on all of these transcripts, and therefore $\phi_{i^*}^{(i^*)} = 1$.

□

*Claim* 6. From knowledge extraction, we have, $\mathsf{Hyb}_a$ and $\mathsf{Hyb}_b$ are computationally indistinguishable and $\mathsf{Hyb}_z$ and $\mathsf{Hyb}_y$ are computationally indistinguishable.

*Proof.* We run the bounded-depth extractor one-by-one. Let $\mathsf{SubHyb}_i$ be such that we run the extractor on commitments $c_j$ for $j \in \{1, \ldots, i\}$. Additionally, in this sub-hybrid the verifier uses the low-depth verification to compute values $\phi_j^{(i^*)}$ for every $j \in \{1, \ldots, i\}$.

Clearly $\mathsf{SubHyb}_0 = \mathsf{Hyb}_a$ and $\mathsf{SubHyb}_n = \mathsf{Hyb}_b$ are identical hybrids.

Between any two adjacent sub-hybrids, $\mathsf{SubHyb}_{j'-1}$ and $\mathsf{SubHyb}_{j'}$, let the challenger compute value $v'_{j'}$ during the finalize phase and compute flag, $\phi_{j'}^{(i^*)}$.

There are two cases (note that the randomness is only over the challenger's random coins as the adversary is deterministic),

- The value $\phi_{j'}^{(i^*)} = 0$ in sub-hybrid $j'$.

  We must have computed $\phi_{j'}^{(i^*)} = 0$ in sub-hybrid $j' - 1$ as the output of the brute force verification and the fast parallel verification is the same.

  Since the challenger computes $\phi_{j'}^{(i^*)} = 0$ in both sub-hybrids, the challenger computes the same $\Omega_{i^*}$ and the probability the adversary outputs 1 in the final experiment is the same.

[15] Recall that in the non-uniform model, we can derandomize any adversary by fixing its random coins to the choice that maximizes the adversary's advantage; this fixed set of coins is in turn provided to the adversary as *advice*.

- The value $\phi_{j'}^{(i^*)} = 1$ in sub-hybrid $j'$.

  We must have computed $\phi_{j'}^{(i^*)} = 1$ in sub-hybrid $j' - 1$ as the output of the brute force verification and the fast parallel verification is the same.

  In this case, from Definition 17, the extractor extracts the same value $v'_{j'}$ in both sub-hybrids with some negligible loss. Thus the value computed $\Omega$st only differs in negligible probability.

Thus, between adjacent sub-hybrids, the probability that the experiments differ is negligible. The final indistinguishability between $\mathsf{Hyb}_a$ and $\mathsf{Hyb}_b$ holds by a standard hybrid argument. Similarly, we can say that $\mathsf{Hyb}_z$ and $\mathsf{Hyb}_y$ are identical. □

*Claim* 7. From zero knowledge simulation, we have that $\mathsf{Hyb}_b$ and $\mathsf{Hyb}_c$ are computationally indistinguishable to bounded depth adversaries. The same holds for $\mathsf{Hyb}_y$ and $\mathsf{Hyb}_x$.

*Proof.* We present the argument that $\mathsf{Hyb}_b$ and $\mathsf{Hyb}_c$ are computationally indistinguishable; a symmetric argument applies to $\mathsf{Hyb}_y$ and $\mathsf{Hyb}_x$.

We'll break the argument down into $n$ subhybrids $\mathsf{SubHyb}_0 = \mathsf{Hyb}_b$ and $\mathsf{SubHyb}_n = \mathsf{Hyb}_c$. In $\mathsf{SubHyb}_i$, the first $i$ proofs by $P_{i^*}$ are replaced with the zero-knowledge simulator. That is, instead of prover $P_{i^*}$ engaging in its interactive proofs with verifying parties $P_1, \ldots, P_i$, $P_{i\mathrm{st}}$ is replaced with the zero-knowledge simulator.

We show that any two subhybrids $\mathsf{SubHyb}_i$, $\mathsf{SubHyb}_{i+1}$ are computationally indistinguishable. Recall that the protocols involve a preprocessing phase before $(\mathsf{pp}_{i^*}, c_{i^*})$ is revealed. After $(\mathsf{pp}_{i^*}, c_{i^*})$ is revealed, the remainder of the protocol can be run in bounded depth: it involves only running the bounded-depth extractor, and generating and verifying proofs. As stated in Section 3.6, we required that these operations are computable in bounded depth.

Therefore, if these hybrids are distinguishable by some adversary $\mathcal{B}$, there exists an adversary $(\mathcal{A}_0, \mathcal{A}_1)$ breaking zero-knowledge as follows. $\mathcal{A}_0$ simulates $\mathcal{B}$ in the preprocessing phase of the protocol, runs preprocessing on behalf of the honest prover $P_{i\mathrm{st}}$, and runs preprocessing for all simulators $\mathcal{S}_0$. $\mathcal{A}_1$ takes as input advice generated by $\mathcal{B}$, along with advice for the honest prover and simulators. It uses $\mathcal{S}_1$ to simulate the honest prover's interaction with $P_1, \ldots, P_i$. For party $P_{i+1}$, it uses its zero-knowledge challenger's protocol run as $P_{i^*}$'s behavior. At the end of this interaction, it outputs the guess of $\mathcal{B}$.

Observe that if $\mathcal{A}_1$ was interacting with the actual prover, this is exactly $\mathsf{SubHyb}_i$; if it was interacting with the simulator, this is $\mathsf{SubHyb}_{i+1}$. Therefore, if $\mathcal{B}$ succeeds in distinguishing between these two hybrids, $\mathcal{A}$ succeeds in distinguishing between the simulator and the actual protocol execution. Since $\mathcal{B}$ is bounded-depth once the input of the zero-knowledge proof is specified (i.e., the start of the interactive phase), $\mathcal{A}_1$ is bounded-depth.

□

*Claim* 8. From hiding of the commitment, we have, $\mathsf{Hyb}_c$ and $\mathsf{Hyb}_x$ are computationally indistinguishable.

*Proof.* Consider an adversary $(\mathcal{A}_0, \mathcal{A}_1)$ distinguishing between these two hybrids. We'll construct $(\mathcal{B}_0, \mathcal{B}_1)$ that break the fine-grained commitment scheme.

$\mathcal{B}_0$ simulates $\mathcal{A}_0$ and obtains its advice string $\alpha$. $\mathcal{B}_0$ outputs messages $m_0 = 0$ and $m_1 = 1$, and sends $\alpha$ to $\mathcal{B}_1$. $\mathcal{B}_1$ receives the challenge $c$, which is either $\mathsf{Com}(\mathsf{pp}, 0)$ or $\mathsf{Com}(\mathsf{pp}, 1)$. It then simulates the protocol in $\mathsf{Hyb}_c$ and $\mathsf{Hyb}_x$ where $P_{i^*}$'s contribution is $c$, interacting with the distinguisher $\mathcal{A}_1$. That is, $\mathcal{B}_1$ uses the zero-knowledge simulator for $P_{i^*}$'s proofs, and derives the outputs using the extractor. $\mathcal{B}_1$ then outputs $\mathcal{A}_1$'s output.

Observe that $\mathcal{B}_1$ runs in parallel time at most $O(t(\lambda))$. Furthermore, if $c$ is $\mathsf{Com}(\mathsf{pp}, 0)$ this simulation is exactly $\mathsf{Hyb}_c$; if $c$ is $\mathsf{Com}(\mathsf{pp}, 1)$ this is exactly $\mathsf{Hyb}_y$. Therefore, $\mathcal{B}_1$ breaks hiding. □

Combining the claims above, we have that, $\mathsf{Hyb}_0$ is computationally indistingusihable from $\mathsf{Hyb}_1$. Let $\beta = \oplus_{j \neq i^*} \phi_j^{(i^*)} \cdot v'_j$. Thus, we have that,

$$|\Pr[\Omega' = 1 \mid \Omega_{i^*} = \beta] - \Pr[\Omega' = 1 \mid \Omega_{i^*} = 1 \oplus \beta]|$$

is negligible. Since the commitment scheme is perfectly binding, i.e. there is a unique value $v'_j$ for $j \neq i^*$ (if there is no value then $\phi_j^{(i^*)} = 0$), and is fixed during $\mathsf{Preprocess}$ (before calling adversary $\mathcal{A}_1$), in the experiment, $\beta$ is either 0 or 1. Thus, we have, $|\Pr[\Omega' = 1 \mid \Omega_{i^*} = 0] - \Pr[\Omega' = 1 \mid \Omega_{i^*} = 1]|$ is negligible. □

**Acknowledgments.** We thank Yuval Efron for helpful discussions and Kecheng Shi for pointing an error in the modified Dolev Strong protocol in an earlier version of the manuscript.

Marshall Ball is supported in part by the National Science Foundation CAREER Award #2443735. Miranda Christ was partially supported by a Google CyberNYC grant, an Amazon Research Award, and NSF grants CCF-2312242, CCF-2107187, and CCF-2212233. This work was done in part while some of the authors were visiting the Simons Institute for the Theory of Computing. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation.

# References


[AMZ24] Shweta Agrawalr, Giulio Malavolta, and Tianwei Zhang. Time-lock puzzles from lattices. In *Annual International Cryptology Conference*, pages 425–456. Springer, 2024.

[ARS24] Damiano Abram, Lawrence Roy, and Mark Simkin. Time-based cryptography from weaker assumptions: Randomness beacons, delay functions and more. *Cryptology ePrint Archive*, 2024.

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[BBCE25] Joseph Bonneau, Benedikt Bünz, Miranda Christ, and Yuval Efron. Good things come to those who wait - dishonest-majority coin-flipping requires delay functions. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VII*, volume 15607 of *Lecture Notes in Computer Science*, pages 225–253. Springer, 2025.

[BG97] Mihir Bellare and Shafi Goldwasser. Verifiable partial key escrow. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97: 4th Conference on Computer and Communications Security*, pages 78–91, Zurich, Switzerland, April 1–4, 1997. ACM Press.

[BKP18] Nir Bitansky, Yael Tauman Kalai, and Omer Paneth. Multi-collision resistance: a paradigm for keyless hash functions. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 671–684, Los Angeles, CA, USA, June 25–29, 2018. ACM Press.

[Blu82] Manuel Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137, 1982.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer Berlin Heidelberg, Germany.

[CATB23] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. In *Financial Crypto*, 2023.

[CCB24] Miranda Christ, Kevin Choi, and Joseph Bonneau. Cornucopia: Distributed Randomness at Scale. In *Advances in Financial Technologies*, 2024.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *TOC*, 1986.

[CMB23] Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 75–92. IEEE, 2023.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147, Santa Barbara, CA, USA, August 16–20, 1993. Springer Berlin Heidelberg, Germany.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[FKPS21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 447–479, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA*, pages 25–32. ACM, 1989.

[GM24] Jiaxin Guan and Hart Montgomery. On sequential functions and fine-grained cryptography. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part V*, Lecture Notes in Computer Science, pages 393–428. Springer, 2024.

[Gol01] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*, volume 2. Cambridge university press, 2001.

[GS98] David M Goldschlag and Stuart G Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In *Financial Crypto*, 1998.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. 1979.

[LHW+22] Hsun Lee, Yuming Hsu, Jing-Jie Wang, Hao Cheng Yang, Yu-Heng Chen, Yih-Chun Hu, and Hsu-Chun Hsiao. HeadStart: Efficiently Verifiable and Low-Latency Participatory Randomness Generation at Scale. In *NDSS*, 2022.

[LW17] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*, pages 620–649. Springer, 2019.

[RSW96] RL Rivest, A Shamir, and DA Wagner. Time-lock puzzles and timed-release crypto, 1996.

[SJH+23] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In *NDSS*, 2023.