CaSCaDE: (Time-Based) Cryptography from Space Communications DElay

Carsten Baum[1⋆], Bernardo Machado David[2⋆⋆]
Elena Pagnin[3⋆⋆⋆], and Akira Takahashi[4†]

[1] Technical University of Denmark, cabau@dtu.dk
[2] IT University of Copenhagen, bernardo@bmdavid.com
[3] Chalmers University of Technology, elenap@chalmers.se
[4] J.P.Morgan AI Research & AlgoCRYPT CoE, takahashi.akira.58s@gmail.com

July 5, 2024

Abstract. Time-based cryptographic primitives such as Time-Lock Puzzles (TLPs) and Verifiable Delay Functions (VDFs) have proven to be pivotal in several areas of cryptography. All existing candidate constructions, however, guarantee time-delays based on the average hardness of sequential computational problems. This means that any algorithmic or hardware improvement affects parameter choices and may turn deployed systems insecure.

To address this issue, we investigate how to build time-based cryptographic primitives where delays depend on sources other than sequential computations: namely, *transmission delays caused by sequential communication*. We explore sequential communication delays that arise when sending a message through a constellation of satellites in *Space*. This setting has the advantage that distances between protocol participants are *guaranteed* as positions of satellites are *observable* from Earth, moreover *delay lower bounds* are unconditional and can be easily computed using the laws of *Physics* (no transmission travels faster than the speed of Light).

We introduce proofs of sequential communication delay (SCD) in the *Universal Composability framework*, that can be used to convince a verifier that a message has accrued delay by traversing a path among a set of scattered satellites. With our SCD proofs we realize the first proposals of Publicly Verifiable TLPs and VDFs whose *delay guarantees are rooted on physical limits, rather than ever-decreasing computational hardness.* Finally, our notion of SCD paves the way to the first Delay Encryption construction not based on supersingular isogenies.

⋆ This work was supported by Protocol Labs.
⋆⋆ This work was supported by Protocol Labs and the Independent Research Fund Denmark (IRFD) grant number 0165-00079B.
⋆⋆⋆ This work was supported by the VR project number 2022-04684.
† Work partially done while affiliated with the University of Edinburgh.

Table of Contents

1 Introduction

Time-Lock Puzzles (TLPs) [42] and Verifiable Delay Functions (VDFs) [11] have received a lot of attention recently as building blocks for *e.g.* randomness beacons and multiparty computation with partial fairness. The minimum delay in evaluating a VDF or solving a TLP is obtained by forcing parties to solve computational problems that require a number of inherently sequential steps. Even if the hardness of sequential computational problems is well understood in theory, lower bounds for the *concrete* time spent computing a number of sequential steps heavily depends on the (evolving) algorithms and hardware used for such computation.

In this work, we take a different approach and investigate how to construct time-based cryptographic primitives from *physical assumptions* (in addition to classical trust assumption). This allows us to build proofs of sequential communication delay based on Physics phenomena that have strong experimental evidence and are absolute. Our constructions derive their delay guarantees from special relativity, which posits that communication cannot happen faster than the speed of Light. Thus, the communication delay between two parties is precisely lower bounded by their relative distance. In detail, let d denote the distance in meters between two satellites, and c the speed of Light, then the minimal possible time-delay in their communication is $\Delta = d/c$ (see Appendix E for more information). The fundamental physical delay Δ becomes apparent when transmitting data over large distances of thousands of km. This motivates the use of communication delay incurred by sending messages across a constellation of satellites placed far from each other, in Space.

Practical Considerations About Our Setting. The advent of relatively cheap CubeSats [41] has made it possible to easily deploy sizable constellations for specific applications, sparking initiatives towards satellite-based cryptographic applications [1]. It is therefore not unthinkable that satellite time could be rented from different satellite providers in the future, similar to how one rents cloud servers today. Moreover, having satellites owned by different companies participating also shows that an assumed threshold on corruptions in a constellation of satellites is realistic. Using satellites also allows any third party verifiers to ascertain communication delay lower bounds. There are many ways to track satellites and learn their positions, which allows for determining communication delay lower bounds as discussed in Appendix E. Finally, since it is hard and costly to tamper with a satellite in orbit, one can combine the physical delay guarantees of special relativity with computational delay guarantees provided by an on-board non-programmable computational device (*e.g.* an ASIC) that solves hard sequential problems (*e.g.* iterated squaring) with well-known runtimes. Since it is infeasible to update the internal device, later advances that speed up these runtimes do not affect the on-board computation.

Can't These Delays Be Simulated Locally? Since we also rely on a trust assumption (for reasons that will be more clear later), it might seem that delays can be

trivially guaranteed by having honest parties quarantine messages for a certain period of time before transmitting them, instead of relying on physical delay. Such a naive solution exploiting local delays would require *tightly* synchronized clocks, which are expensive to realize in practice. Moreover, the trivial solution does not provide precise and absolute delay lower bounds because clocks on real-world devices may be susceptible to unexpected physical faults. We are thus motivated to propose a better solution that requires synchronization only to guarantee termination, while achieving the minimum delays imposed by the speed-of-light limit and large distances, even if an honest party has *loosely synchronized clocks*. We aim to derive absolute delay lower bounds that hold even if an adversary completely controls the clock. In contrast, such an adversary can cheat honest parties of the trivial protocol into believing that a certain minimum delay was guaranteed.

Related Work. Time-Lock Puzzles (TLPs) [42] allow a sender to commit to a message in such a way that a receiver can obtain it only after a delay is elapsed. Verifiable Delay Functions (VDFs) [11] work as a pseudorandom function whose evaluation requires at least a certain delay, after which it generates both an output and a proof that the output was obtained after this delay. Similarly, a publicly verifiable TLP (PV-TLP) also produces a proof that a certain message was contained in the puzzle. In both cases, verifying these proofs takes time essentially independent of the delay for solving the PV-TLP or evaluating the VDF. A lot of theoretical work has been done on constructing TLPs [8, 9, 13, 26, 30, 42] and VDFs [7, 11, 24, 25, 39, 46]. Yet, all known constructions are based on the average hardness of sequential computational problems, and hence are orthogonal to this work.

The concept of deriving security guarantees based on physical assumptions is not new in the field of cryptography, e.g., noisy communication channels [21, 21, 36], physically-unclonable functions [14, 38, 43], tamper-proof tokens [28, 29], and more recently protein polymers for secure vaults and one-time programs [3]. None of the aforementioned assumptions, however, enforces time delays. We proceed along the line of work – initiated by Kent [32] in 1999 – that builds cryptographic schemes from special relativity. In detail [32, 34] focus on commitments, more recent efforts target multi-prover Zero-Knowledge proofs [22] and have been experimentally demonstrated [2, 45]. However, these constructions require verifiers to interact with provers via ideal secure channels, whereas the primitives we consider require non-interactive public verifiability.

1.1 Our Contributions

We build on the line of work that builds cryptographic schemes from special relativity assumptions and provide the following contributions.

Modelling dynamic delayed channels in UC: We introduce a UC model for communication channels that incorporate time-varying delays. We model both single-use channels and multiple-use channels. To achieve this, we utilize a Global clock to establish synchrony. Since our model accounts for messages

being transmitted through a constellation of satellites, it accurately captures the communication delay between parties whose positions change over time. This variability in position directly impacts the delay experienced when transmitting messages between these parties.

Proofs of Sequential Communication Delay: Building on our model, we introduce techniques for proving that a certain message has been sequentially transmitted among a number of parties. We analyse the delay bounds obtained by composing delayed channels and propose the notion of proofs of sequential communication delay (SCD). We propose SCD protocols based on physical delays, digital signatures, and PKI and prove them secure in the UC-hybrid model.

VDF from SCD: We present the first construction of a UC-secure VDF based on physical communication delay. Specifically, we construct VDFs from proofs of sequential communication delay and a bulletin board, in the random oracle model, by extracting randomness from such proofs.

TLP from SCD: We present the first construction of a UC-secure publicly verifiable (PV) time-lock puzzle based on physical communication delay. As an application, we show that our PV-TLPs can be used to efficiently instantiate the randomness beacon from [7] (expensive resources are only used in case of cheating).

Delay Encryption and Stateless VDF from Threshold Identity Based Encryption (IBE) and SCD: We show how to obtain Delay Encryption [15] by combining our proofs of sequential communication delay and an IBE scheme endowed with a threshold identity secret key generation protocol. To the best of our knowledge, this is the first delay encryption scheme not based on supersingular isogeny assumptions. We also use a similar technique to obtain a more efficient construction of VDFs.

1.2 Technical Overview

We briefly describe the new models and techniques we introduce to obtain our results. In Appendix E we elaborate on our model choices, lower bounds on the communication delay, and performance estimates of the core building blocks.

Modelling Communication Delays: We model time and delays by using a global clock $\mathcal{G}_{\mathsf{Clock}}$ inspired by [33], which we realize in the Abstract Composable Time [8] framework. We start by modelling a single-use ideal functionality $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ for delayed message transmission with fixed minimum ($\Delta_{\mathtt{lo}}$) and maximum ($\Delta_{\mathtt{hi}}$) delay parameters. This functionality guarantees that the adversary does not learn the message until at least $\Delta_{\mathtt{lo}}$ ticks after it is sent by the sender. At the same time, the functionality guarantees that the receiver gets the message at most $\Delta_{\mathtt{hi}}$ ticks after sending. This means that as long as at least one of the channel participants is honest, the channel guarantees delay. Next, we model a multiple use delayed channel functionality $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ where the minimum and maximum delays for a message sent at time t are dynamically determined by a function $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$ according to the current time provided by $\mathcal{G}_{\mathsf{Clock}}$. We show that $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ can be realized based on $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$. These functionalities can

be composed to obtain a minimum delay guarantee for a message transmitted among multiple parties.

Proofs of SCD: We define the notion of a proof of sequential communication delay $\pi_{\mathtt{lo}}$, which allows a third party verifier $\mathcal{V}$ to check that a given message m has been sent from $\mathcal{P}_S$ to $\mathcal{P}_R$ while incurring a minimum delay of $\Delta_{\mathtt{lo}}$. This is modelled using the functionality $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. As we construct this functionality from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, we inherit security guarantees of minimal delay even if one of the participants (sender or receiver) is dishonest. We then construct a simple protocol realizing $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ in a synchronized setting with a global clock $\mathcal{G}_{\mathsf{Clock}}$, public key infrastructure $\mathcal{F}_{\mathsf{Reg}}$, a unique digital signature scheme $\mathcal{F}_{\mathsf{Sig}}$ and a delayed channel $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$. In this protocol, $\mathcal{P}_S$ signs (m, t), *i.e.* the message to be sent concatenated with the time t when the message is sent, obtaining a signature $\sigma_{\mathcal{P}_S}$. $\mathcal{P}_S$ sends $(m, t, \sigma_{\mathcal{P}_S})$ through $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ to $\mathcal{P}_R$, who checks that $\sigma_{\mathcal{P}_S}$ is valid w.r.t. (m, t) and $\mathcal{P}_S$'s verification key. Moreover, $\mathcal{P}_R$ checks that it has received the message at a time t' such that $t + \Delta_{\mathtt{lo}} \leq t' \leq t + \Delta_{\mathtt{hi}}$, where $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$, *i.e.* it checks that the claimed sending time t is consistent with the channels parameters and the time t' when it actually receives the message. If the checks pass, $\mathcal{P}_R$ generates a signature $\sigma_{\mathcal{P}_R}$ on $(m, t, \sigma_{\mathcal{P}_S})$, and a proof $\pi_{\Delta_{\mathtt{lo}}}^{\mathcal{P}_S \to \mathcal{P}_R} = (\sigma_{\mathcal{P}_S}, \sigma_{\mathcal{P}_R})$ along with $m, t, \Delta_{\mathtt{lo}}$. Any third party can verify whether the proof is valid w.r.t. a message m and parameters $t, \Delta_{\mathtt{lo}}$ by checking that $\mathcal{P}_S, \mathcal{P}_R$ are the parties transmitting through $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, that $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$ and that $\sigma_{\mathcal{P}_S}, \sigma_{\mathcal{P}_R}$ are valid. $\pi_{\mathtt{lo}}$ gives no guarantees if both $\mathcal{P}_S$ and $\mathcal{P}_R$ collude, *e.g.* by sharing their signing keys; we show that $\pi_{\mathtt{lo}}$ is a proof of delay if at most one of the parties is corrupted. This simple protocol can be generalized to a chain of delay channels with multiple intermediate parties. Our definition of $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ is broad enough that this functionality can also model this setting. Here, the parameters of the individual delay channels of the participating parties, their number as well as the threshold of parties that can be corrupted allow us to prove which delay function $\mathsf{f}_\Delta(\cdot)$ the sequential proof will guarantee.

VDF from SCD: We obtain a direct construction of a VDF from proofs of sequential communication delay. Our construction departs from $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, a random oracle and a bulletin board $\mathcal{F}_{\mathsf{BB}}$, which is used to keep the state of VDF evaluations. The core idea is to send the VDF input in from a sender $\mathcal{P}_S$ to a receiver $\mathcal{P}_R$ via $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, obtaining a proof of sequential communication delay $\pi_{\mathtt{lo}}$, which is also the proof of VDF evaluation. The output of the VDF is determined by querying the random oracle on $in|\pi_{\mathtt{lo}}$. Verification can be done by checking $\pi_{\mathtt{lo}}$ is valid for a given minimal delay $\Delta_{\mathtt{lo}}$ and recomputing the output. The first $\pi_{\mathtt{lo}}$ is written to the bulletin board and retrieved for future evaluations of the VDF to avoid multiple valid evaluations (*i.e.* sending in through $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ again to get a different $\pi_{\mathtt{lo}}$).

Delay Encryption and Stateless VDF from Threshold IBE and SCD: We use an IBE scheme with a protocol that allows parties who hold shares of the master secret key to efficiently generate the secret key for a given identity. In order to construct Delay Encryption, we define encryption under a given identity as IBE encryption under that identity. Later on, key extraction is done

by having parties jointly generate the secret key for that identity in a round-robin manner along with a proof of sequential communication delay showing that this key generation had a certain minimum delay. We then adapt this technique to obtain a unique signature using Naor's transform from IBE to signatures, which yields a threshold unique signature with a proof of sequential communication delay attesting the minimum delay for signature generation. The final VDF can be constructed by applying a random oracle to the signature and using the signature of communication delay and the signature itself to verify the VDF output. This construction solves the caveat of our first simple construction, since it always yields the same output for each input without requiring any parties to keep state.

TLP from SCD: We construct a PV-TLP protocol where a puzzle is a ciphertext puz obtained by encrypting a message m under the public key pk of a threshold encryption scheme. The parties $\mathcal{P}_i$ who have the corresponding secret key shares sk_i are connected via delayed channels $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$. A PV-TLP is solved by threshold decrypting puz via delayed channels $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ following a specific sequence of parties $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\}$ where $\mathcal{P}_i$ aggregates its decryption share to $\mathcal{P}_{i-1}$'s decryption share before passing it on to $\mathcal{P}_{i+1}$. The delay guarantee comes from our analysis of sequential communication delay, as honest parties $\mathcal{P}_i$ check that the ciphertext has traversed the path from $\mathcal{P}_1$ to $\mathcal{P}_{i-1}$ before aggregating their decryption share, which guarantees a minimum delay. In order to obtain a publicly verifiable proof that a puzzle puz contained a message m, we employ the random oracle based transformation of [27, 40], where decryption yields not only m but the unique randomness used to generate puz. This randomness constitutes our proof, since together with m it can be used to do a re-encryption check.

2 Preliminaries

Notation. We denote the computational (resp. statistical) security parameter by τ (resp. λ), the concatenation of two strings a and b by $a|b$, and compact multiple concatenations by $(a_i)_{i=1}^n = a_1|a_2|\ldots|a_n$.

Auxiliary Background Material. In Appendix A, we give an overview of the UC framework [17] and present standard functionalities for Public Key Infrastructures ($\mathcal{F}_{\mathsf{Reg}}$), (unique) digital signatures ($\mathcal{F}_{\mathsf{Sig}}$), bulletin boards ($\mathcal{F}_{\mathsf{BB}}$) and global random oracles ($\mathcal{G}_{\mathsf{rpoRO}}$), which we will use in our constructions.

UC Secure Public-Key Encryption with Plaintext Verification. It is observed in [6] that it is possible to UC-realize public-key encryption with a plaintext verification property using the random oracle-based IND-CCA secure public-key encryption schemes of [27, 40]. This plaintext verification property allows a party who decrypts a ciphertext to generate a non-interactive publicly verifiable proof that a certain plaintext was obtained. We will apply the approach of [6] to obtain a threshold public-key encryption scheme with the same plaintext verification property. In order to do so, we use the fact that the encryption

schemes of [27,40] can be obtained from any partially trapdoor one-way function, which allows us to depart from a simple threshold version of El Gamal to obtain a UC-secure theshold encryption scheme with plaintext verification. In Appendix A.1 we recall in verbatim form the definitions of the schemes from [27, 40] and the necessary properties for obtaining plaintext verification as presented in [6].

Global Clocks. We need to assume that honest parties have synchronized clocks. [5] This is necessary to argue about evolving communication delays with respect to specific instants in time, which we need to construct proofs of sequential communication delays. We capture this notion of synchronicity by using a global clock functionality $\mathcal{G}_{\mathsf{Clock}}$,following the ideas of [4,31,33]. $\mathcal{G}_{\mathsf{Clock}}$ allows parties and functionalities to request the current value of a synchronized time counter, which is only incremented if all honest parties agree to update the clock. This also means that *e.g.* ticks cannot happen randomly in protocol steps, unless parties in the protocol explicitly query $\mathcal{G}_{\mathsf{Clock}}$ to continue. We explain in Appendix A.2 how $\mathcal{G}_{\mathsf{Clock}}$ can be realized in the framework of [8].

Functionality 1: $\mathcal{G}_{\mathsf{Clock}}$

$\mathcal{G}_{\mathsf{Clock}}$ is parameterized by a variable ν, sets $\mathcal{P}, \mathcal{F}$ of parties and functionalities respectively. It keeps a Boolean variable $d_{\mathcal{J}}$ for each $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$, a counter ν as well as an additional variable $\mathtt{u}$. All $d_{\mathcal{J}}$, ν and $\mathtt{u}$ are initialized as 0.

Clock Update: Upon receiving a message (UPDATE) from $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$: Set $d_{\mathcal{J}} = 1$. If $d_F = 1$ for all $F \in \mathcal{F}$ and $d_p = 1$ for all honest $p \in \mathcal{P}$, set $\mathtt{u} \leftarrow 1$ if it is 0.

Clock Read: Upon receiving a message (READ) from any entity: If $\mathtt{u} = 1$ then first send (TICK, sid) to $\mathcal{S}$. Next set $\nu \leftarrow \nu + 1$, reset $d_{\mathcal{J}}$ to 0 for all $\mathcal{J} \in \mathcal{P} \cup \mathcal{F}$ and reset $\mathtt{u}$ to 0. Answer the entity with (READ, ν).

3 Modeling Communication Delays

We model physical communication between two parties as authenticated message transmission ideal functionalities that ensure both minimal and maximal communication delays. This is in line with communication in the UC framework, that always happens through channel functionalities. Moreover, we allow any third party to observe the minimum and maximum delay bounds for a message transmitted through the functionality. This implicitly assumes that the parties know each others' positions (in order to compute the delays) which is a reasonable assumption for satellites and base stations as outlined in the introduction.

[5] Our protocols in fact only require loosely synchronized clocks, as the minimum delay is guaranteed by a physical effect rather than synchronization, and the use of synchronization only impacts liveness of the protocol. We choose not to model that more explicitly as it would require more details in the formalization.

Functionality 2: $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$

This functionality is parameterized by a minimal delay $\Delta_{\mathtt{lo}} > 0$ and a maximal delay $\Delta_{\mathtt{hi}} > \Delta_{\mathtt{lo}}$; it interacts with a sender $\mathcal{P}_S$, a receiver $\mathcal{P}_R$, an adversary $\mathcal{S}$, and the clock $\mathcal{G}_{\mathsf{Clock}}$. At initialisation t is set to 0, and the flags msg, $\mathsf{released}$, done to $\bot$.

Send: Upon receiving an input $(\textsc{Send}, \mathsf{sid}, m)$ from party $\mathcal{P}_S$, do:
- If $\mathsf{msg} = \bot$, record m and set $\mathsf{msg} = \top$.
- If $\mathsf{msg} = \top$, send $(\textsc{None}, \mathsf{sid})$ to $\mathcal{P}_S$.

Receive: Upon receiving $(\textsc{Rec}, \mathsf{sid})$ from $\mathcal{P}_R$, do:
- If $\mathsf{released} = \bot$ and $\mathsf{done} = \bot$, then send $(\textsc{None}, \mathsf{sid})$ to $\mathcal{P}_R$.
- If $\mathsf{released} = \top$ and $\mathsf{done} = \bot$, then $\mathsf{msg} = \top$ and there exists a recorded message m. Set $\mathsf{done} = \top$ and send $(\textsc{Sent}, \mathsf{sid}, m)$ to $\mathcal{P}_R$.
- If $\mathsf{done} = \top$, then send $(\textsc{done}, \mathsf{sid})$ to $\mathcal{P}_R$.

Release message: Upon receiving an input $(\textsc{ok}, \mathsf{sid})$ from $\mathcal{S}$, do:
- If $\mathsf{msg} = \bot$ or $t < \Delta_{\mathtt{lo}}$, then send $(\textsc{None}, \mathsf{sid})$ to $\mathcal{S}$.
- If $\mathsf{msg} = \top$, $t \geq \Delta_{\mathtt{lo}}$ and $\mathsf{released} = \bot$, then set $\mathsf{released} = \top$.
- If $\mathsf{released} = \top$, then send $(\textsc{None}, \mathsf{sid})$ to $\mathcal{S}$.

Tick: Sends $(\textsc{Read})$ to $\mathcal{G}_{\mathsf{Clock}}$, receiving $(\textsc{Read}, \bar{t})$ as answer. If $\bar{t}$ has changed since the last activaction:
- If $\mathsf{msg} = \bot$, then send $(\textsc{None}, \mathsf{sid})$ to $\mathcal{S}$.
- If $\mathsf{msg} = \top$ and $\mathsf{released} = \bot$, then set $t = t + 1$:
  - If $t = \Delta_{\mathtt{lo}}$ then send $(\textsc{Sent}, \mathsf{sid}, m, t)$ to $\mathcal{S}$.
  - If $t = \Delta_{\mathtt{hi}}$, set $\mathsf{released} = \top$ and send $(\textsc{Released}, \mathsf{sid})$ to $\mathcal{S}$.

We start by modelling a single-use delayed channel with fixed minimum and maximum delay parameters for simplicity. This channel captures the transmission of a single message between two parties at an specific point in time, which determines the delay parameters. As parties' relative positions evolve with time, so do the communication delay bounds as their relative distances change. We therefore, based on the single-use delayed channel, construct a multi-use functionality whose delay bounds can evolve with the ticks of $\mathcal{G}_{\mathsf{Clock}}$. This multi-use channel allows other parties to observe the delay bounds for a message transmitted at a given (past or future) point in time, which will later be necessary for verifying the output of a time-based primitive constructed over the channels, as well as estimating the delay guaranteed by a future evaluation of such a primitive.

Single-Use Channel ideal functionality $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$: As a warm-up example, we present the functionality $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ for delayed authenticated message transmission in Functionality 2. The message delivery is at least $\Delta_{\mathtt{lo}}$ ticks (*i.e.* the physical bound for message transmission), and this delay holds also for an adversarial receiver. The adversary cannot force transmission to be delayed by more than $\Delta_{\mathtt{hi}}$ ticks if it is the sender, and cannot force delivery before $\Delta_{\mathtt{lo}}$ ticks.

Multiple-Use Channel ideal functionality $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$: Manually keeping track of what instance of $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ to use (along with its parameters $\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}$) every time a message needs to be sent between two parties, as well as the current time, would make protocol descriptions very cumbersome. Hence, we present a higher level abstraction of a multiple-use delayed authenticated channel that

Functionality 3: $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$

This functionality is parameterized by a computational security parameter τ and a permissible delay function $\mathsf{f}_\Delta : \{0, \ldots, \mathsf{poly}(\tau)\} \to \mathbb{N} \times \mathbb{N}$; it interacts with $\mathcal{G}_{\mathsf{Clock}}$, sender $\mathcal{P}_S$, receiver $\mathcal{P}_R$ and adversary $\mathcal{S}$. At initialisation the list L is empty.
In any call below, $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ first sends (READ) to $\mathcal{G}_{\mathsf{Clock}}$ and obtains $(\text{READ}, \bar{t})$.

Send: Upon first message $(\text{SEND}, \mathsf{sid}, m)$ for $\bar{t}$ from party $\mathcal{P}_S$ add $(m, \bar{t}, \bot)$ to L.
Receive: Upon receiving $(\text{REC}, \mathsf{sid})$ from $\mathcal{P}_R$, for every $(m, t, \mathsf{released}) \in L$, if $\mathsf{released} = \top$ (*i.e.* the maximum delay has passed or the adversary released the message), remove $(m, t, \mathsf{released})$ from L and send $(\text{SENT}, \mathsf{sid}, m, t)$ to $\mathcal{P}_R$.
Release message: Upon receiving an input $(\text{OK}, \mathsf{sid}, t)$ from $\mathcal{S}$ compute $(\Delta_{\mathtt{lo}}, \cdot) \leftarrow \mathsf{f}_\Delta(t)$. If there is $(m, t, \mathsf{released}) \in L$ such that $\bar{t} \geq t + \Delta_{\mathtt{lo}}$ then set $\mathsf{released} = \top$.
Tick: For every $(m, t, \mathsf{released}) \in L$ compute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$ and do as follows:
- If $t + \Delta_{\mathtt{lo}} = \bar{t}$, send $(\text{SENT}, \mathsf{sid}, m, t)$ to $\mathcal{S}$.
- If $t + \Delta_{\mathtt{hi}} = \bar{t}$, set $\mathsf{released} = \top$.

automatically assigns minimum and maximum delays to each message according to the time it is sent. In Functionality 3 we present the functionality $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ for multiple-use delayed authenticated message transmission. The main parameter of this functionality is a function f_Δ that takes as input a time t and outputs the minimum delay $\Delta_{\mathtt{lo}}$ and maximum delay $\Delta_{\mathtt{hi}}$ for a message sent at time t. When it is requested to transmit a message, $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ determines the current time by contacting $\mathcal{G}_{\mathsf{Clock}}$ and computes $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$. Next, the functionality registers the message in a list and ensures that it is not revealed to the adversary before a minimum delay $\Delta_{\mathtt{lo}}$, while guaranteeing delivery to an honest receiver within a maximum delay $\Delta_{\mathtt{hi}}$. Moreover, $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ allows for any third party to obtain the delay parameters for messages sent at a given clock tick, as f_Δ is a public parameter of the functionality (similar to $\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}$ in $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}}$).

To model predictability of delay, we require that the variance between any two ticks in delay - as modeled by f_Δ - cannot be too much: no adversary should be able to send a message *faster by waiting until a later tick* (i.e. time travel of messages is not possible). To capture this, we give the following definition:

Definition 1 (Permissible Delay Function). *A function* $f_\Delta : \{0, \ldots, \mathsf{poly}(\tau)\} \to \mathbb{N} \times \mathbb{N}$ *models permissible delay if*

$$\forall t \in \mathbb{N} : (\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow f_\Delta(t), (\Delta'_{\mathtt{lo}}, \Delta'_{\mathtt{hi}}) \leftarrow f_\Delta(t+1) \Rightarrow \Delta'_{\mathtt{lo}} - \Delta_{\mathtt{lo}} > -1.$$

Realizing $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ *from* $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}}$ In Appendix B.1 we present a protocol that realises the multiple-use ideal functionality for authenticated delayed message transmission using $\mathcal{G}_{\mathsf{Clock}}$ and multiple $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}}$, and prove its security. The protocol uses one instance of $\mathcal{F}_{\mathsf{dmt}}^{\cdot}$ for each possible timestamp. The sender simply picks the correct instance for message transmission, while the verifier for every clock tick tests 1. if any of the instances delivers a message to him; and 2. if the message's time of sending and delay are consistent.

4 Proofs of Sequential Communication Delays

In this section, we introduce techniques for producing a publicly verifiable proof $\pi_{\mathtt{lo}}$ that a message m has incurred a certain minimum delay due to being transmitted from party $\mathcal{P}_S$ to party $\mathcal{P}_R$. Such a proof, when using the delay channel functionalities from Section 3, requires that at least one of the two parties involved in the process was honest. The idea is to have both the sender $\mathcal{P}_S$ and receiver $\mathcal{P}_R$ of a delayed channel sign the input message and the initial timestamp when this message was sent (provided that the message is received within reasonable time constraints such that the initial timestamp is not too far in the future or past). Both signatures and the initial timestamp form the proof $\pi_{\mathtt{lo}}$ showing that the message was sent from $\mathcal{P}_S$ to $\mathcal{P}_R$ incurring a given minimum delay as observed by an honest party. This is guaranteed by the delayed channel, whose minimum delay is determined by the timestamp.

We then use a sequence of consecutive communication channels between multiple parties in order to obtain a larger provable minimum delay than that provided by a single channel without intermediaries. Here, a message m travels from sender $\mathcal{P}_1$ to receiver $\mathcal{P}_n$, through hops $\mathcal{P}_i$. Each intermediate party $\mathcal{P}_i$ sends to $\mathcal{P}_{i+1}$ not only the original message m, but also a proof showing that m travelled from $\mathcal{P}_1$ to $\mathcal{P}_i$. If m and the proof arrive at $\mathcal{P}_i$ at a certain time that is not consistent with the minimum and maximum delays of the channels connecting $\mathcal{P}_1$ to $\mathcal{P}_i$ (*i.e.* it is too far in the future or in the past), $\mathcal{P}_i$ aborts. This construction can be leveraged to obtain a final proof of sequential communication delay consisting of $(m, t, \sigma_1, \ldots, \sigma_{i-1})$, where signature σ_i is generated by party $\mathcal{P}_i$, and t is the initial timestamp when m was sent. Finally, we sketch the optimization that uses sequentially aggregate signatures (SAS) [35] and ordered multi signatures (OMS) [10] to avoid proofs of sequential communication delay of size linear in the number of network nodes.

4.1 Modelling Proofs of Sequential Communication Delay

We begin by modeling a publicly verifiable proof of delay through an ideal functionality $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ depicted in Functionality 4. This functionality incorporates the delayed channel modelled by $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, and proof generation/verification mechanisms similar to those of the unique digital signature functionality $\mathcal{F}_{\mathsf{Sig}}$ (Functionality 13). Departing from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, which allows for a $\mathcal{P}_S$ to send a message m to $\mathcal{P}_R$ with minimum and maximum delays $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$ depending on time t, $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ delivers to $\mathcal{P}_R$ the proof $\pi_{\mathtt{lo}}$ that m was sent at time t with a minimum delay $\Delta_{\mathtt{lo}}$.

In $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, the adversary may only generate valid proofs of delay after the minimal delay of $\pi_{\mathtt{lo}}$, but it learns m earlier than the honest receiver. This makes sense because the statement that the message m has traveled for a certain delay does not mean that m was only learnt by the adversary with that delay. For an example in practice, consider a chain of 4 parties with 3 intermediate delay channels. If e.g. $\mathcal{P}_2$ and $\mathcal{P}_4 = \mathcal{P}_R$ are both corrupted, then the adversary must of

Functionality 4: $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$

$\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ keeps initially empty lists L, L_π, and is parameterized by a computational security parameter τ and a permissible delay function f_Δ. $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ interacts with $\mathcal{G}_{\mathsf{Clock}}$, sender $\mathcal{P}_S$, receiver $\mathcal{P}_R$, verifiers $\mathcal{V}$ and adversary $\mathcal{S}$.

In any call below, $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ first sends (READ) to $\mathcal{G}_{\mathsf{Clock}}$ and obtains $(\text{READ}, \bar{t})$.

Send: Upon receiving an input $(\text{SEND}, \mathsf{sid}, m)$ from an honest $\mathcal{P}_S$ and if this is the first such message in this tick-round:

1. Compute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(\bar{t})$ and add $(\bar{t}, m, \bot, \Delta_{\mathtt{lo}})$ to L.
2. Output $(\text{MESSAGE}, \mathsf{sid}, \bar{t}, m)$ to $\mathcal{S}$.

If $\mathcal{P}_S$ is corrupted, then upon input $(\text{SEND}, \mathsf{sid}, m, t)$ from $\mathcal{S}$ compute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$. If $\Delta_{\mathtt{hi}} + t - \bar{t} \geq \Delta_{\mathtt{lo}}$ then add $(t, m, \bot, \Delta_{\mathtt{lo}})$ to L.

Receive: Upon receiving $(\text{REC}, \mathsf{sid})$ from $\mathcal{P}_R$, for every $(t, m, \top, \mathtt{cnt}) \in L$:

1. Remove $(t, m, \mathsf{released}, \mathtt{cnt})$ from L and recompute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$.
2. If $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi_{\mathtt{lo}}, 1) \in L_\pi$ send $(\text{SENT}, \mathsf{sid}, m, t, \bar{t} - t, \pi_{\mathtt{lo}})$ to $\mathcal{P}_R$.
3. Else, send $(\text{PROOF}, \mathsf{sid}, m, t, \bar{t} - t)$ to $\mathcal{S}$. Upon receiving $(\text{PROOF}, \mathsf{sid}, m, t, \pi_{\mathtt{lo}})$ from $\mathcal{S}$, check that $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi_{\mathtt{lo}}, 0) \notin L_\pi$. If yes, output $\bot$ to $\mathcal{P}_S/\mathcal{P}_R$ and halt. Else, add $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi_{\mathtt{lo}}, 1)$ to L_π and send $(\text{SENT}, \mathsf{sid}, m, t, \bar{t} - t, \pi_{\mathtt{lo}})$ to $\mathcal{P}_R$. If $\mathcal{S}$ sends $(\text{NOPROOF}, \mathsf{sid})$ then output $(\text{NOPROOF}, \mathsf{sid})$.

Release message: Upon receiving an input $(\text{OK}, \mathsf{sid}, t)$ from $\mathcal{S}$ compute $(\Delta_{\mathtt{lo}}, \cdot) \leftarrow \mathsf{f}_\Delta(t)$. If there is $(t, m, \mathsf{released}, \mathtt{cnt}) \in L$ such that $\bar{t} \geq t + \Delta_{\mathtt{lo}}$ and $\mathtt{cnt} = 0$ then set $\mathsf{released} = \top$.

Verify: Upon receiving $(\text{VERIFY}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}})$ from $\mathcal{V}_i$, send $(\text{VERIFY}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}})$ to $\mathcal{S}$. Upon receiving $(\text{VERIFIED}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, \phi)$ from $\mathcal{S}$ do:

1. If $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$ and $\Delta \notin [\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}]$ or then set $f = 0$. Otherwise set $f = 1$. (is delay in allowed interval?)
2. If $t + \Delta_{\mathtt{lo}} > \bar{t}$ then set $f = 0$ (no verification request can be positive, unless m has circulated for at least $\Delta_{\mathtt{lo}}$ ticks)
3. If $\phi = 1$ and there is an entry $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi'_{\mathtt{lo}}, 1) \in L_\pi$ where $\pi'_{\mathtt{lo}} \neq \pi_{\mathtt{lo}}$ then set $f = 0$. (any proof of delay must be unique)
4. If there is an entry $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi_{\mathtt{lo}}, f') \in L_\pi$, let $b = f \wedge f'$. (All verification requests with identical parameters will result in the same answer.)
5. If no such entry is present, set $b = f \wedge \phi$ and add $(m, t, \Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}, \pi_{\mathtt{lo}}, b)$ to L_π. (Add for consistency)

Output $(\text{VERIFIED}, \mathsf{sid}, m, t, \Delta, b)$ to $\mathcal{V}_i$.

Tick: For every $(t, m, \mathsf{released}, \mathtt{cnt}) \in L$ compute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$. If $t + \Delta_{\mathtt{hi}} = \bar{t}$, set $\mathsf{released} = \top$. If $\mathtt{cnt} > 0$ then reduce $\mathtt{cnt}$ by 1.

course learn m once it arrives at $\mathcal{P}_2$. The guarantee of the functionality is that *the proof of delay* will only arrive at the adversary with the required minimal delay, and that an *honest* receiver will have to potentially wait longer to receive it and m. This is because the message still has to pass through channels that have honest parties as senders and receivers before a proof is generated.

A second interesting property is that a corrupted sender is allowed to *date back* message sending by a certain amount of ticks, i.e. at time $\bar{t}$ it is allowed to say that it sent the message already at time $t < \bar{t}$. It can do so as long as $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ can still delay proof (and message) delivery by $\Delta_{\mathtt{lo}}$ ticks without exceeding time $t + \Delta_{\mathtt{hi}}$ during delivery. The reason for this "time traveling" of dishonest

senders is the multiparty protocol. For example, consider a chain of parties where $\mathcal{P}_1 = \mathcal{P}_S$ and $\mathcal{P}_2$ are corrupted. In that case the simulator cannot extract any information from the channel between $\mathcal{P}_1$ and $\mathcal{P}_2$ as the adversary is of course not bound to use this channel. But it can still guarantee message delay as parties later on in the chain are honest, so their delay channels must have been used.

A third important property is that a proof that m was sent through the channel with a certain delay that is within $[\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}]$ is *unique* to the tuple (m, t), where t is the time when m was supposed to be sent. Moreover, $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ allows any verifier $\mathcal{V}_i$ to check that a proof $\pi_{\mathtt{lo}}$ of delay in $[\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}]$ for message m sent at time t is indeed valid (*i.e.* it has been generated honestly). Here, the adversary may define validity of a proof during verification even if $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ did not output the proof itself at that time. This is an artifact of our protocol, as a dishonest receiver $\mathcal{P}_R$ must not make his contributions public until when the proof gets verified. This is standard behavior in other UC functionalities, such as the signature functionality $\mathcal{F}_{\mathsf{Sig}}$.

Proofs of Sequential Communication Delay with 2 parties. We construct a simple protocol (Protocol 5) that realizes $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ between two parties by leveraging a delayed channel $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, a Public Key Infrastructure $\mathcal{F}_{\mathsf{Reg}}$ and a unique digital signature $\mathcal{F}_{\mathsf{Sig}}$ on a synchronized network (with synchrony maintained by $\mathcal{G}_{\mathsf{Clock}}$). In it, both the sender $\mathcal{P}_S$ and receiver $\mathcal{P}_R$ sign the message m being transmitted. However, we need to take steps to guarantee that an honest $\mathcal{P}_R$ does not inadvertently help a corrupted $\mathcal{P}_S$ forge a proof for an invalid initial timestamp t or minimum delay $\Delta_{\mathtt{lo}}$. In order to to avoid this issue, $\mathcal{P}_R$ needs to verify that m has been received through an instance of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ where $\mathcal{P}_S$ acts as sender at a timestamp between $t + \Delta_{\mathtt{lo}}$ and $t + \Delta_{\mathtt{hi}}$, where t is the initial timestamp when the message was sent and $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_{\boldsymbol{\Delta}}(t)$. Since $\mathcal{P}_R$ needs to know t in order to obtain $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}})$, we have $\mathcal{P}_S$ sign (m, t), allowing $\mathcal{P}_R$ to perform its delay consistency checks. If $\mathcal{P}_R$ is satisfied, it then signs (m, t, σ_S), where σ_S is $\mathcal{P}_S$'s signature, and outputs both $\mathcal{P}_S$'s signature and its own as the proof of sequential communication delay. Verifying such a proof of sequential communication delay can be done by any third party by verifying the signatures generated $\mathcal{P}_S$ and $\mathcal{P}_R$, as well as checking consistency of the timestamps.

Theorem 1. *π_{SCD} (Protocol 5) UC-realizes $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ in the $\mathcal{G}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}, \mathcal{F}_{\mathsf{Sig}}, \mathcal{F}_{\mathsf{Reg}}$-hybrid model against a static active adversary corrupting at most one of $\mathcal{P}_S, \mathcal{P}_R$.*

The proof can be found in Appendix B.2 and is rather straightforward. For a corrupted sender, extract the message m from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ but ensure that verification keys are registered and that it would later be accepted by an honest receiver. For a corrupted receiver, program $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ to output the correctly signed message at the right time. In this case, verification is more involved as upon querying **Verify** the signature used by the dishonest receiver might be undefined.

Proofs of Sequential Communication Delay with > 2 parties. We will now realize $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ using a *longer chain of parties*. There, the sender $\mathcal{P}_1 = \mathcal{P}_S$ is connected to $\mathcal{P}_2$ using a delayed channel $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ with delay function $\mathsf{f}_{\Delta,1}$, $\mathcal{P}_2$ is connected to $\mathcal{P}_3$

Protocol 5: π_{SCD}

Protocol π_{SCD} is executed by a sender $\mathcal{P}_S$, a receiver $\mathcal{P}_R$ and a set of verifiers $\mathcal{V}$ interacting with each other and with $\mathcal{G}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}, {\mathcal{F}_{\mathsf{Sig}}}^S, {\mathcal{F}_{\mathsf{Sig}}}^R, \mathcal{F}_{\mathsf{Reg}}$.
In every step, the activated party sends (READ) to $\mathcal{G}_{\mathsf{Clock}}$ to obtain $(\text{READ}, \bar{t})$.

Setup: Upon first activation, party $\mathcal{P}_i \in \{\mathcal{P}_S, \mathcal{P}_R\}$ proceeds as follows:
1. Send $(\text{KEYGEN}, \mathsf{sid})$ to an instance of ${\mathcal{F}_{\mathsf{Sig}}}^i$ where it acts as signer;
2. Upon receiving $(\text{VERIFICATION KEY}, \mathsf{sid}, \mathsf{SIG}.vk_i)$ from ${\mathcal{F}_{\mathsf{Sig}}}^i$, $\mathcal{P}_i$ sends $(\text{REGISTER}, \mathsf{sid}, \mathsf{SIG}.vk_i)$ to $\mathcal{F}_{\mathsf{Reg}}$.

Send: Upon receiving first input $(\mathsf{Send}, \mathsf{sid}, m)$ for $\bar{t}$, $\mathcal{P}_S$ proceeds as follows:
1. Send $(\text{SIGN}, \mathsf{sid}, (m, \bar{t}))$ to ${\mathcal{F}_{\mathsf{Sig}}}^S$, receiving $(\text{SIGNATURE}, \mathsf{sid}, (m, \bar{t}), \sigma_S)$.
2. Send $(\text{SEND}, \mathsf{sid}, (m, \bar{t}, \sigma_S))$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$.

Receive: Upon receiving $(\text{REC}, \mathsf{sid})$, $\mathcal{P}_R$ sends $(\text{REC}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ and proceeds as follows for every $(\text{SENT}, \mathsf{sid}, (m, t, \sigma_S), t')$ received from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$:
1. Check that $t = t'$ and $\mathtt{verifySigs}(\mathcal{P}_\mathtt{S}, (\mathtt{m}, \mathtt{t}), \sigma_\mathtt{S}, \mathtt{t})$ evaluates to true.
2. If the checks pass, send $(\text{SIGN}, \mathsf{sid}, (m, t, \sigma_S))$ to ${\mathcal{F}_{\mathsf{Sig}}}^R$, receiving $(\text{SIGNATURE}, \mathsf{sid}, (m, t, \sigma_S), \sigma_R)$. Output $(\text{SENT}, \mathsf{sid}, m, t, \bar{t} - t, (\sigma_S, \sigma_R))$.
3. If a check fails, then output $(\text{NOPROOF}, \mathsf{sid})$.

Verify: Upon receiving $(\text{VERIFY}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}})$, $\mathcal{V}_i \in \mathcal{V}$ parses $\pi_{\mathtt{lo}} = (\sigma_S, \sigma_R)$ and proceeds as follows:
1. Compute $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t)$. Check that $\Delta \in [\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}]$ and $\bar{t} \geq t + \Delta_{\mathtt{lo}}$.
2. Check that $\mathtt{verifySigs}(\mathcal{P}_\mathtt{S}, (\mathtt{m}, \mathtt{t}), \sigma_\mathtt{S}, \mathtt{t})$ and $\mathtt{verifySigs}(\mathcal{P}_\mathtt{R}, (\mathtt{m}, \mathtt{t}, \sigma_\mathtt{S}), \sigma_\mathtt{R}, \mathtt{t})$ both evaluate to true.
3. If all checks pass set $b = 1$, else $b = 0$. Output $(\text{VERIFIED}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, b)$.

Tick: Send (UPDATE) to $\mathcal{G}_{\mathsf{Clock}}$.

Function $\mathsf{verifySigs}(\mathcal{P}_i, m, \sigma, t)$**:**
1. Send $(\text{RETRIEVE}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{Reg}}$, receiving $(\text{RETRIEVE}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SIG}.vk, t_{Reg})$ as answer. Check that $t_{Reg} \leq t$ and output false if not.
2. Send $(\text{VERIFY}, \mathsf{sid}, m, \sigma, \mathsf{SIG}.vk)$ to ${\mathcal{F}_{\mathsf{Sig}}}^i$, receiving $(\text{VERIFIED}, \mathsf{sid}, m, \sigma, f)$ as response. Output true if $f = 1$, otherwise false.

via $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ with $\mathsf{f}_{\Delta,2}$ until $\mathcal{P}_{n-1}$, which is connected via $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ to $\mathcal{P}_n = \mathcal{P}_R$ with delay function $\mathsf{f}_{\Delta,n}$. As before, $\mathcal{P}_1$ signs m, t before sending it through $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, while $\mathcal{P}_2$ signs the output of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ if it is valid and then forwards it with the signature via $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ to $\mathcal{P}_3$ etc. We will prove that such a chain again realizes an instance of $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, but with different delay parameters.

We consider malicious adversaries that can *interrupt signature generation* by refusing to execute the protocol. We assume that each party in the chain knows all the delay functions $\mathsf{f}_{\Delta,i}$ for each of the $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ instances in the chain, which allows them to compute delay bounds for incoming messages. In our protocol, $\mathcal{P}_i$ must establish that the message m that it obtained – which was supposedly initially sent at time t by $\mathcal{P}_1$ – *could be delivered to* $\mathcal{P}_{i-1}$ via instances of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ with delay functions $\mathsf{f}_{\Delta 1}, \ldots, \mathsf{f}_{\Delta i-2}$ and incurring the respective delay, such that $\mathcal{P}_{i-1}$ sending it at time t_{i-1} via $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ with a delay modeled by $\mathsf{f}_{\Delta i-1}$ is plausible.

As an example, assume a chain of 3 parties where only $\mathcal{P}_3$ is honest. Let $(1, 3) = \mathsf{f}_{\Delta,1}(t) = \mathsf{f}_{\Delta,2}(t)$ for every t, and assume that $\mathcal{P}_3$ obtains m from $\mathcal{P}_2$, which was supposedly sent at $t = 0$ by $\mathcal{P}_1$. $\mathcal{P}_3$ knows that m must travel a

minimum time of 1 tick from $\mathcal{P}_1$ to $\mathcal{P}_2$ or at most 3 ticks. If the channel from $\mathcal{P}_2$ to $\mathcal{P}_3$ incurs delay between 1 and 3 ticks, but $\mathcal{P}_3$ obtains m at tick 7, then $\mathcal{P}_2$ has sent m the earliest at tick 4. This means that $\mathcal{P}_2$ is cheating as it delayed delivery of m. Alternatively, if $\mathcal{P}_3$ had obtained m at tick 1 then $\mathcal{P}_2$ must have sent the message at tick 0 (by the minimum delay $\mathsf{f}_{\Delta,2}$), which is also impossible as the message would have needed at least 1 tick from $\mathcal{P}_1$ to $\mathcal{P}_2$. Hence, we carefully specify *what each party verifies before signing* about timestamps and delivery times and how it impacts the proven delay given the corruption thresholds.

Plausible delays. We now introduce the $\mathsf{plausible}$ delay predicate $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,\ell-1}, t_\ell)$. It is defined for $\ell > 1$ as follows:

$\ell = 2$**:** true if $\exists \Delta \in \mathsf{f}_{\Delta,1}(t_1) : t_1 + \Delta = t_2$.
$\ell > 2$**:** true if $\exists \Delta \in \mathsf{f}_{\Delta,1}(t_1)$: $\mathsf{isP}(t_1 + \Delta, \mathsf{f}_{\Delta,2}, \ldots, \mathsf{f}_{\Delta,\ell-1}, t_\ell)$.

As we constrain the output of each $\mathsf{f}_{\Delta,i}$ to only be defined on polynomially many inputs, isP can be computed in polynomial time as long as $\ell = O(\log(\tau))$. This can be improved if, e.g. all f_{Δ} functions are constant in an obvious way.

We now show that we can combine two instances of isP into one:

Proposition 1. *Let* $\mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}$ *be permissible delay functions and let* t_1, t_i, t_n *be such that* $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,i-1}, t_i)$ *and* $\mathsf{isP}(t_i, \mathsf{f}_{\Delta,i}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n)$*. Then* $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n)$ *holds.*

Conversely, we can also decompose every isP chain into its parts.

Proposition 2. *Let* $\mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}$ *be permissible delay functions and* t_1, t_n *be such that* $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n)$ *holds. For every* $i \in \{2, \ldots, n-1\}$ *there exists a* t_i *such that* $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,i-1}, t_i)$ *and* $\mathsf{isP}(t_i, \mathsf{f}_{\Delta,i}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n)$ *hold.*

Proof. (of Propositions 1 & 2) The definition of isP implies that $\mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n)$ returns true if and only if $\exists\ \Delta_1, \ldots, \Delta_{n-1} : t_1 + \sum_{i=1}^{n-1} \Delta_i = t_n \wedge \Delta_i \in \mathsf{f}_{\Delta,i}\left(t_1 + \sum_{j=1}^{i-1} \Delta_j\right)$. Proposition 1 follows by combining both existential statements. Proposition 2 follows from setting $t_{i+1} = t_1 + \Delta_1 + \cdots + \Delta_i$. □

We stress that verifying that a message *arrived at the receiver with plausible delay* does not imply that it indeed incurred the delay during delivery. The reason for this is that if a sequence of parties are corrupted, then they may not use delayed channels for communication among each other. Going back to the aforementioned example, if m arrives at tick-round 2 at $\mathcal{P}_3$ and is claimed to have been sent at tick round $t = 0$ by $\mathcal{P}_1$, then this is not what must have happened as we first must consider the corruption threshold. If both $\mathcal{P}_1, \mathcal{P}_2$ are corrupted then an adversary could have only gotten m at tick round 1, signed $(m, 0)$ using both signing keys and make $\mathcal{P}_2$ send it to $\mathcal{P}_3$. Hence, if we consider a corruption model where 2 parties out of 3 can be corrupted, the overall channel built by $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ cannot guarantee a minimum delay that is longer than 1, if by minimum delay we mean time spent for m to travel as observed by honest parties. This is of course different if only 1 out of $\mathcal{P}_1, \mathcal{P}_2$ can be corrupted.

We now describe how the proven minimal delivery time can be computed. If both $\mathcal{P}_1$ & $\mathcal{P}_n$ are honest, then $\mathcal{P}_n$ would only sign if isP is true when the message

arrives at it. This means that the message must have incurred a delay from $\mathcal{P}_1$ to $\mathcal{P}_n$ that is at least the sum of minimal delays on each intermediate channel: $\mathcal{P}_1$ is honest and must have sent it at the right time. Therefore, the longest chain of delay observed by the honest parties in this case spans the whole message delay from $\mathcal{P}_1$ to $\mathcal{P}_n$ and is the lower-bound on provable message delay. This observation extends to any chain between the first $\mathcal{P}_i$ and last $\mathcal{P}_j$ honest party within $\mathcal{P}_1, \ldots, \mathcal{P}_n$, if either of $\mathcal{P}_1, \mathcal{P}_n$ was not honest. Therefore, to determine the minimal guaranteed delay in case of k corruptions, we only need to consider the cases *where all of* $\mathcal{P}_1, \ldots, \mathcal{P}_{i-1}$ *are dishonest* and send the message later than allowed, or *where* $\mathcal{P}_{j+1}, \ldots, \mathcal{P}_n$ *are all dishonest* and sign the messages earlier than allowed, or both. Only these can reduce proven delay time.

Next, consider the setting where honest parties appear in sequences of at least $n - k > 1$ consecutive parties in the network, i.e. there is no isolated honest party. Let $\mathcal{P}_i, \ldots, \mathcal{P}_{i+n-k-1}$ be such an honest chain of parties. Then the minimal delay cannot be reduced by placing a dishonest party within this chain. This follows because then either $\mathcal{P}_{i-1}$ or $\mathcal{P}_{i+n-k}$ become honest, and the minimal honest delay then consists of the minimal delay on $\mathcal{P}_i, \ldots, \mathcal{P}_{i+n-k-1}$ plus the extra party (as the additional delay due to f_Δ will be non-negative. Therefore, to reduce the minimal delay to a minimum, exactly $n-k$ consecutive parties must be honest.

Moreover, it is not sufficient if only $\mathcal{P}_1$ or $\mathcal{P}_n$ is dishonest, followed or preceded by honest parties. This is because an honest $\mathcal{P}_2$ by observing $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ would ensure that the message was sent early enough given the delay of the channel (similarly for an honest $\mathcal{P}_{n-1}$ and corrupt $\mathcal{P}_n$). Thus, to minimize delay, an adversary will not only corrupt the first or last party in the chain, but also the adjacent one.

Bounding the channel delays. Using Propositions 1, 2 and the aforementioned observations, we can compute the minimal and maximal delay by decomposing an isP sequence into all possible partitions of up to 3 plausible subsequences, one of which is of length $n - k$ and represents the honest parties. There are at most $\mathsf{poly}(\tau)$ many such decompositions. In Appendix B.3 we show how to find sequences that realize the shortest observable minimal delay, or the maximal delay, in time polynomial in the number of isP calls.

Putting things together. We present a detailed description of our protocol for sequential communication delays $\pi_{\mathsf{Multi-SCD}}$ in Protocol 18 in Appendix B. The protocol realizes the delay function delays computable as outlined previously.

Theorem 2. *The protocol* $\pi_{\mathsf{Multi-SCD}}$ *UC-securely implements* $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ *in the* $\mathcal{G}_{\mathsf{Clock}}$, $\mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{Sig}}, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$*-hybrid model with security against any adversary actively corrupting up to* $k = n - 1$ *parties with permissible delay function given by* delays.

The proof can be found in Appendix B.4 and follows a similar outline as the one for Theorem 1. The key difference is that there might be a dishonest $\mathcal{P}_S$, followed by a chain of dishonest $\mathcal{P}_2, \mathcal{P}_3, \ldots$ that do not necessarily have to communicate via their $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ instances. Hence, when the first honest (simulated) party obtains an output from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$, then the message that $\mathcal{S}$ enters into $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ has to have an earlier timestamp than the current one, based on the claim when the dishonest $\mathcal{P}_1$ originally "sent" the message.

Optimizing $\pi_{\mathsf{Multi-SCD}}$ While $\pi_{\mathsf{Multi-SCD}}$ realizes $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ using only simple primitives, it incurs a large overhead for the proof of sequential communication: one proof consists of n nested signatures, and each party $\mathcal{P}_i$ forwards i signatures to party $\mathcal{P}_{i+1}$. We want to obtain a proof size and communication complexity independent from the number of parties, preferably close to the size of a single signature. To do so, we face a main hurdle: it seems that we cannot eliminate a signature by any intermediate party, since that would allow the adversary to forge proofs by making the eliminated party be the honest party in $\pi_{\mathsf{Multi-SCD}}$. Hence, we focus on techniques that allow us to aggregate signatures by each party $\mathcal{P}_i$ involved in $\pi_{\mathsf{Multi-SCD}}$ in such a way that we obtain a compact proof of size independent from n. A conceptually simple way to achieve this is using a sequentially aggregate signature scheme (SAS) [35] or an ordered multi-signatures scheme (OMS) [10], which allow for aggregating a number of signatures generated in sequence into a single signature (*i.e.* with the same size as a single signature). This directly fits our use of signatures in $\pi_{\mathsf{Multi-SCD}}$, where enforcing the order of signing is solved by the SAS/OMS property of allowing verifiers to check the order with which each party generated its signature on (m, t).

5 Verifiable Delay Functions

We construct a VDF from proofs of sequential communication delays. Our construction can be obtained in a black-box manner from any proof of sequential delay, yielding a VDF with a proof size equal to that of the underlying proof of communication delay. The main idea is to sequentially send the input of the VDF among nodes in a network while having them compute a proof of sequential communication delay for this message. The output is computed by querying a global random oracle on the input concatenated with the proof of sequential communication delay. Verification can be easily achieved by first verifying the proof of sequential communication delay and then recomputing the output. We realize a VDF functionality (dapted from [7]) presented in Functionality 6.

We present our VDF protocol in Protocol 7. The construction assumes access to a bulletin board where we store attempts at jointly evaluating the VDF by sending a message via $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. When evaluating the VDF we consider as valid only the first evaluation attempt registered in the bulletin board with a valid proof of sequential delay generated by $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. This significantly simplifies our analysis since the adversary can no longer send the same input to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ multiple times and obtain multiple proofs of sequential delay and thus produce several valid VDF outputs, which deviates from the standard behavior expected from this primitive. The same effect could be obtained by assuming either $\mathcal{P}_S$ or $\mathcal{P}_R$ are honest and do not accept to interact with $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ to transmit the same message more than once, thus guaranteeing only one proof of sequential delay is generated, which means a single valid VDF output exists.

Theorem 3. *Protocol* π_{VDF} *UC-realizes* $\mathcal{F}_{\mathsf{VDF}}$ *in the* $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}, \mathcal{G}_{\mathsf{rpoRO}}, \mathcal{F}_{\mathsf{BB}}$*-hybrid model against an active static adversary corrupting a majority of parties in* $\mathcal{P}$*.*

Functionality 6: $\mathcal{F}_{\mathsf{VDF}}$

$\mathcal{F}_{\mathsf{VDF}}$ is parameterized by a computational security parameter τ, and input space $\mathcal{ST}$, a proof space $\mathcal{PROOF}$, a slack parameter $0 < \epsilon \leq 1$ and a delay parameter Γ. $\mathcal{F}_{\mathsf{VDF}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{VDF}}$ maintains a initially empty lists L (proofs being computed), and OUT (outputs).

Solve: Upon receiving $(\mathsf{Solve}, \mathsf{sid}, in)$ from $\mathcal{P}_i \in \mathcal{P}$ where $in \in \mathcal{ST}$ and $\Gamma \in \mathbb{N}$, add $(\mathcal{P}_i, \mathsf{sid}, in, 0, \top)$ to L and send $(\mathsf{Solve}, \mathsf{sid}, in)$ to $\mathcal{S}$.

Tick: For each $(\mathcal{P}_i, \mathsf{sid}, in, c, b) \in L$, update $(\mathcal{P}_i, \mathsf{sid}, in, c, b) \in L$ by setting $c = c+1$ and proceed as follows:

1. If $c \geq \epsilon\Gamma$ sample $out \xleftarrow{\$} \mathcal{ST}$, send $(\mathsf{GetStsPf}, \mathsf{sid}, in, out)$ to $\mathcal{S}$ and wait for an answer. If $\mathcal{S}$ answers with $(\textsc{Abort}, \mathsf{sid})$, update $(\mathcal{P}_i, \mathsf{sid}, in, c, b) \in L$ by setting $b = \bot$. If $\mathcal{S}$ answers with $(\mathsf{GetStsPf}, \mathsf{sid}, \pi)$, $\mathcal{F}_{\mathsf{VDF}}$ halts if $\pi \notin \mathcal{PROOF}$ or there exists $(in', out', \pi) \in \mathsf{OUT}$, else, it appends (in, out, π) to OUT.
2. If $c = \Gamma$, remove $(\mathcal{P}_i, \mathsf{sid}, in, \Gamma, b) \in L$. If there was an abort (*i.e.* $b = \bot$), send $(\mathsf{NoProof}, \mathsf{sid}, in)$ to $\mathcal{P}_i$. Otherwise, send $(\mathsf{Proof}, \mathsf{sid}, in, out, \pi)$ to $\mathcal{P}_i$.

Verification: Upon receiving $(\mathsf{Verify}, \mathsf{sid}, in, out, \pi)$ from $\mathcal{P}_i \in \mathcal{P}$, set $b = 1$ if $(in, out, \pi) \in \mathsf{OUT}$, otherwise set $b = 0$ and output $(\mathsf{Verified}, \mathsf{sid}, in, out, \pi, b)$ to $\mathcal{P}_i$.

The delay parameter is $\Gamma = \Delta_{\mathtt{hi}}$ *and the slack parameter is* $\epsilon = \frac{\Delta_{\mathtt{lo}}}{\Delta_{\mathtt{hi}}}$ *where* $(\cdot, \Delta_{\mathtt{hi}}) = \max_{t \in \{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$ *and* $(\Delta_{\mathtt{lo}}, \cdot) = \min_{t \in \{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$.

Proof. It is simple to construct a simulator $\mathcal{S}$ for π_{VDF} by having $\mathcal{S}$ interact with an internal copy of $\mathcal{A}$ towards which it simulates honest parties executing exactly as in π_{VDF} and simulating $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}, \mathcal{G}_{\mathsf{rpoRO}}, \mathcal{F}_{\mathsf{BB}}$ exactly as they are described except when explicitly stated. $\mathcal{S}$ forwards every message sent to simulated $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ to be evaluated by $\mathcal{F}_{\mathsf{VDF}}$ and provides matching proofs to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ and $\mathcal{F}_{\mathsf{VDF}}$ when requested. If $\mathcal{A}$ causes an evaluation to abort, $\mathcal{S}$ correspondingly aborts the same evaluation at $\mathcal{F}_{\mathsf{VDF}}$. Whenever $\mathcal{F}_{\mathsf{VDF}}$ leaks to $\mathcal{S}$ that an evaluation on a new input has been requested, $\mathcal{S}$ simulates this evaluation in the simulation. Moreover, $\mathcal{S}$ programs $\mathcal{G}_{\mathsf{rpoRO}}$ so that outputs of simulated VDF evaluations match the outputs provided by $\mathcal{F}_{\mathsf{VDF}}$. □

6 Delay Encryption

In this section, we extend our PV-TLP construction to obtain a related primitive called Delay Encryption [15]. A Delay Encryption scheme allows for encrypting many messages under a certain identity in such a way that a secret key allowing for decrypting all such messages can be obtained after a certain delay, a notion akin to an "identity based TLP". We construct this primitive by combining an IBE scheme with a distributed (identity) key generation protocol and our proofs of sequential communication delay.

Assume $\mathtt{IBE} = (\mathtt{Setup}, \mathtt{KG}, \mathtt{Enc}, \mathtt{Dec})$ is an Identity-based encryption scheme where: $\mathtt{IBE.Setup}$ on input the security parameter τ outputs the master secret key msk and the public key pk; $\mathtt{IBE.KG}$ on input an identity string $ID \in \{0,1\}^*$ and msk outputs the identity decryption key sk_{ID}; $\mathtt{IBE.Enc}$ on input the plaintext

Protocol 7: π_{VDF}

Protocol π_{VDF} is executed by a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ interacting with a bulletin board functionality $\mathcal{F}_{\mathsf{BB}}$ and with $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, where party $\mathcal{P}_R \in \mathcal{P}$ acts as receiver and party $\mathcal{P}_S \in \mathcal{P}$ as sender. They additionally use a random oracle $\mathcal{G}_{\mathsf{rpoRO}}$.

Solve: A party $\mathcal{P}_i$ interacts with $\mathcal{P}_S, \mathcal{P}_R$ as follows to evaluate the VDF on in:

1. On input $(\mathsf{Solve}, \mathsf{sid}, in)$, $\mathcal{P}_i$ sends $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BB}}$ and checks whether a record $(c, in, t, \Delta, \pi_{1\mathsf{o}})$ is returned (if multiple $(c, in, t, \Delta, \pi_{1\mathsf{o}})$ for different c and $\pi_{1\mathsf{o}}$ are returned, consider the one with the lowest c and a valid $\pi_{1\mathsf{o}}$ w.r.t $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$). If yes, skip to step 5.
2. $\mathcal{P}_i$ sends $(\textsc{Send}, \mathsf{sid}, in)$ to $\mathcal{P}_S$ and $\mathcal{P}_S$ forwards $(\textsc{Send}, \mathsf{sid}, in)$ from $\mathcal{P}_i$ to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$.
3. Upon receiving $(\textsc{Sent}, \mathsf{sid}, in, t, \Delta, \pi_{1\mathsf{o}})$ from $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$, $\mathcal{P}_R$ send $(\textsc{Write}, \mathsf{sid}, (in, t, \Delta, \pi_{1\mathsf{o}}))$ to $\mathcal{F}_{\mathsf{BB}}$. If instead $\mathcal{P}_R$ receives $(\textsc{NoProof}, \mathsf{sid})$, it forwards this message to all parties in $\mathcal{P}$.
4. If it received $(\textsc{NoProof}, \mathsf{sid})$ from $\mathcal{P}_R$, $\mathcal{P}_i$ outputs $(\textsc{NoProof}, \mathsf{sid}, in)$. Otherwise, it sends $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BB}}$ and retrieves $(c, in, t, \Delta, \pi_{1\mathsf{o}})$.
5. $\mathcal{P}_i$ sends $(\textsc{Hash-Query}, in|\pi_{1\mathsf{o}})$ to $\mathcal{G}_{\mathsf{rpoRO}}$, receiving $(\textsc{Hash-Confirm}, out)$. $\mathcal{P}_i$ sends $(\textsc{IsProgrammed}, in|\pi_{1\mathsf{o}})$ and aborts if the response is $(\textsc{IsProgrammed}, 1)$. $\mathcal{P}_i$ outputs $(\textsc{Proof}, \mathsf{sid}, in, out, \pi = \pi_{1\mathsf{o}})$.

Verification: On input$(\textsc{Verify}, \mathsf{sid}, in, out, \pi)$, $\mathcal{P}_i$ proceeds as follows:

1. Send $(\textsc{Read}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BB}}$ and check that there is a record (c, in, t, Δ, π), if multiple (c, in, t, Δ, π) for different c are returned, consider the one with the lowest c and a valid π w.r.t. to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$.
2. Send $(\textsc{Verify}, \mathsf{sid}, in, t, \Delta, \pi)$ to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ expecting $(\textsc{Verified}, \mathsf{sid}, in, t, \Delta, 1)$.
3. Send $(\textsc{Hash-Query}, in|\pi)$ to $\mathcal{G}_{\mathsf{rpoRO}}$, receiving $(\textsc{Hash-Confirm}, out')$. Check that $out = out'$. Send $(\textsc{IsProgrammed}, in|\pi)$ expecting $(\textsc{IsProgrammed}, 0)$.
4. If all checks pass set $b = 1$, else set $b = 0$, and output $(\textsc{Verified}, \mathsf{sid}, in, out, \pi, b)$

m, public key pk and identity ID outputs the ciphertext c; `IBE.Dec` on input the identity decryption key sk_{ID} and the ciphertext c outputs either a message m or $\perp$. First, observe that many IBE schemes (*e.g.* [12]) are essentially a version of El Gamal. This means that `Setup`, `KG` can easily be "thresholdized" to allow for generating identity secret keys from shares of msk, and that sk_{ID} is unique for each ID. As an example, consider [12] which uses two source groups G, a target group G_T and a pairing $e : G \times G \mapsto G_T$. `Setup` creates $pk = g^{msk}$ for master secret key msk using a public generator $g \in G$. `KG` creates a random generator $h = H(ID) \in G$ based on a hash of the identity ID using a random oracle H to G, and lets $sk_{ID} = h^{msk}$. Clearly, sk_{ID} is unique for ID. `Enc` generates a ciphertext $c = (c_1, c_2)$ from m and ID by computing $c_1 = g^r$ $c_2 = m \cdot e(H(ID)^r, pk)$, and `Dec` decrypts c by computing $m = c_2 \cdot e(c_1, sk_{ID})^{-1}$.

It is easy to "thresholdize" such an `IBE` scheme with UC security. To implement `Setup`, parties use standard semi-honest El Gamal distributed key generation to create a Shamir sharing of a random secret msk and then raise g to msk using standard techniques. Additionally, they commit to their shares of msk and

use UC NIZKs to prove execution correctness. Implementing $\mathtt{KG}$ as a distributed protocol is again straightforward as ID is public, since each protocol participant can compute $H(ID)$ locally, raise it to its committed share of msk and prove correctness of this using a UC NIZK. Then, by reconstruction in the exponent, one can obtain the unique $H(ID)^{msk}$. By using a CCA secure version of $\mathtt{Enc}, \mathtt{Dec}$, *e.g.* [12] as shown in [37], we obtain UC security for the full encryption scheme.

Our crucial observation is that we can run a distributed key generation (DKG) protocol outputting the secret key for a given ID via delayed channels $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ that generate proofs of sequential communication. By letting intermediate parties check the key shares and proofs of delay, we can provably lower-bound the delay for creating sk_{ID}. Notice that this idea gives us a natural construction of Delay Encryption. To encrypt, we let a party knowing pk first choose an identity ID and let the ciphertext be $ID, \mathtt{Enc}(m, pk, ID)$. To decrypt one or more ciphertexts for the same ID, parties obtain the secret key sk_{ID} by running the DKG and then decrypt using sk_{ID}. The delay directly follows from the bound on the execution time of the DKG. We provide an ideal functionality for Delay Encryption in Appendix D and formalize this observation in the following theorem, which is conservatively phrased in terms of the [12] IBE, although it can be generalized to any IBE that supports distributed key generation.

Theorem 4. *If the IBE scheme of [12] is* $\mathrm{IND}-\mathrm{ID}-\mathrm{CCA2}$ *secure, there exists a protocol that UC-realizes* $\mathcal{F}_{\mathsf{DE}}$ *in the* $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}, \mathcal{F}_{\mathrm{NIZK}}, \mathcal{G}_{\mathsf{rpoRO}}$*-hybrid model against an active static adversary corrupting a majority of parties in* $\mathcal{P}$*. The delay parameter is* $\Gamma = \Delta_{\mathtt{hi}}$ *and the slack parameter is* $\epsilon = \frac{\Delta_{\mathtt{lo}}}{\Delta_{\mathtt{hi}}}$ *where* $(\cdot, \Delta_{\mathtt{hi}}) = \max_{t \in \{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$ *and* $(\Delta_{\mathtt{lo}}, \cdot) = \min_{t \in \{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$.

7 Publicly Verifiable Time-Lock Puzzles

We construct a publicly verifiable time-lock puzzle (PV-TLP) based on sequential communication delays. The main idea is to use a threshold encryption scheme and generate a puzzle by encrypting a message under the public key. The secret key is in turn shared among a set of nodes connected by delayed channels. The TLP is opened by having these nodes perform threshold decryption via sequential communication. By having the nodes which hold the key shares communicate in a round-robin manner, the individual channel delays then add up to the overall delay of the TLP.

In our construction, the sizes of both the proof and the messages exchanged among each pair of parties involved in solving the puzzle are independent from the number of parties. In order to do so, we relax our output guarantee by only detecting dishonest behavior after the decryption protocol is finished without identifying cheaters, which allows for the adversary to cause aborts without revealing the corrupted parties. In case aborts happen, we can fall back to a more expensive protocol using NIZKs of valid decryption share generation in order to identify the corrupted parties and eliminate them. This yields low overhead in the optimistic case (which is the most likely to happen in practice) while still attaining guaranteed output delivery. See Appendix E for further discussion.

Functionality 8: $\mathcal{F}_{\mathsf{DKG}}$

$\mathcal{F}_{\mathsf{DKG}}$ is parameterized by a cyclic group $\mathbb{G}$ of order q with generator g and interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, among which a subset of solvers $\mathcal{W} \subset \mathcal{P}$.

Key Generation The first time it is activated, $\mathcal{F}_{\mathsf{DKG}}$ samples $\mathsf{sk}_i \xleftarrow{\$} \mathbb{G}$ for $i \in \mathcal{W}$, computes $\mathsf{sk} = \sum_{i \in \mathcal{W}} \mathsf{sk}_i$ and $\mathsf{pk} = g^{\mathsf{sk}}$.
SK Request: Upon $(\textsc{SecKey}, \mathsf{sid})$ from $\mathcal{P}_i \in \mathcal{W}$, return $(\textsc{SecKey}, \mathsf{sid}, \mathsf{sk}_i)$.
PK Request Upon $(\textsc{PubKey}, \mathsf{sid})$ from $\mathcal{P}_i \in \mathcal{P}$, return $(\textsc{PubKey}, \mathsf{sid}, \mathsf{pk})$.

Functionality 9: $\mathcal{F}_{\mathsf{tlp}}$

$\mathcal{F}_{\mathsf{tlp}}$ is parameterized by a computational security parameter τ, a message space $\{0,1\}^\tau$, a tag space $\mathcal{TAG}$, a proof space $\mathcal{PROOF}$, a slack parameter $0 < \epsilon \leq 1$ and a delay parameter Γ. $\mathcal{F}_{\mathsf{tlp}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{tlp}}$ maintains initially empty lists omsg (output messages and proofs) and L (puzzles being solved).

Create puzzle: Upon receiving the first message $(\mathsf{CreatePuzzle}, \mathsf{sid}, m)$ from $\mathcal{P}_i$ where $m \in \{0,1\}^\tau$, proceed as follows:

1. If $\mathcal{P}_i$ is honest, sample $\mathsf{puz} \xleftarrow{\$} \mathcal{TAG}$ and proof $\pi \xleftarrow{\$} \mathcal{PROOF}$.
2. If $\mathcal{P}_i$ is corrupted, let $\mathcal{S}$ provide puz and π. If $(\mathsf{puz}, \pi) \notin \mathcal{TAG} \times \mathcal{PROOF}$ or there exists $(\mathsf{puz}', m', \pi) \in \mathsf{omsg}$, then $\mathcal{F}_{\mathsf{tlp}}$ halts.
3. Append (puz, m, π) to omsg, set and output $(\mathsf{CreatedPuzzle}, \mathsf{sid}, \mathsf{puz}, \pi)$ to $\mathcal{P}_i$ and $(\mathsf{CreatedPuzzle}, \mathsf{sid}, \mathsf{puz})$ to $\mathcal{S}$.

Solve: Upon receiving $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ from $\mathcal{P}_i \in \mathcal{P}$, add $(\mathsf{sid}, \mathsf{puz}, 0)$ to L and send $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ to $\mathcal{S}$.

Public Verification: Upon receiving $(\mathsf{Verify}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ from a party $\mathcal{P}_i \in \mathcal{P}$, set $b = 1$ if $(\mathsf{puz}, m, \pi) \in \mathsf{omsg}$, otherwise set $b = 0$ and output $(\mathsf{Verified}, \mathsf{sid}, \mathsf{puz}, m, \pi, b)$ to $\mathcal{P}_i$.

Tick: For all $(\mathsf{sid}, \mathsf{puz}, c) \in L$, update $(\mathsf{sid}, \mathsf{puz}, c) \in L$ by setting $c = c + 1$ and proceed as follows:

- If $c \geq \epsilon\Gamma$ and $(\mathsf{puz}, m, \pi) \in \mathsf{omsg}$, output $(\mathsf{Solved}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ to $\mathcal{S}$.
- If $c \geq \epsilon\Gamma$ and there does not exist $(\mathsf{puz}, m, \pi) \in \mathsf{omsg}$, let $\mathcal{S}$ provide $\pi \in \mathcal{PROOF}$ and add $(\mathsf{puz}, \bot, \pi)$ to omsg.
- If $c = \Gamma$, remove $(\mathsf{sid}, \mathsf{puz}, c) \in L$ and send $(\mathsf{Proceed?}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ to $\mathcal{S}$, where m, π are such that there is $(\mathsf{puz}, m, \pi) \in \mathsf{omsg}$ and proceed as follows:
  - If $\mathcal{S}$ sends $(\textsc{Abort}, \mathsf{sid}, \pi')$, output $(\mathsf{Solved}, \mathsf{sid}, \mathsf{puz}, \bot, \pi')$ to all $\mathcal{P}_i$.
  - If $\mathcal{S}$ sends $(\textsc{Proceed}, \mathsf{sid})$, output $(\mathsf{Solved}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ to all $\mathcal{P}_i$.

In order to achieve constant communication, we have each decryption node aggregate its decryption share to the share received from the previous party along with a proof of sequential communication showing that the ciphertext being decrypted has traversed a pre-defined path through a certain sequence of decryption nodes. This step avoids attacks where the adversary obtains several decryption shares from different honest nodes in parallel or out of order.

We use the generic Public Key Cryptosystem with Plaintext Verification construction from Definition 5 together with a simple threshold version of El

Gamal to verify that the final decrypted message is indeed the message that was originally encrypted (i.e. the message inside the PV-TLP). Hence, the verifier only has to perform a re-encryption check in order to assert that a given PV-TLP has been correctly solved. This optimized construction realizes the PV-TLP functionality defined in Functionality 9, which follows [7] but supports only a fixed delay Γ. Our construction, $\pi_{\mathsf{TLP-Light}}$, is depicted in Protocol 10 and employs a Distributed Key Generation functionality, $\mathcal{F}_{\mathsf{DKG}}$, in the setup (Functionality 8). The $\mathcal{F}_{\mathsf{DKG}}$ functionality can be UC-realized by a number of protocols that compute a public key g^{sk} and secret key shares sk_i such that $\mathsf{sk} = \mathsf{sk}_1 + \cdots + \mathsf{sk}_n$.

We capture the security of Protocol $\pi_{\mathsf{TLP-Light}}$ in Theorem 5. The proof obtains loose bounds for the minimum and maximum delay guarantees provided by this protocol since $\pi_{\mathsf{TLP-Light}}$ only uses the decryption validity proof as a publicly verifiable proof of a TLP solution, which allows for a unique and easily verifiable proof. If the TLP proof instead also consisted of the proofs provided by the parties in the set $\mathcal{W}$ by using $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ instead of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ and for correct decryption, we would be able to condition the minimum and maximum delays guaranteed by a TLP solution on the exact time when it is solved, which would give tighter delay bounds. However, the latter approach requires an intricate reworking of $\mathcal{F}_{\mathsf{tlp}}$ that would also require a more expensive protocol to realize as the communication per party becomes linear in $|\mathcal{W}|$. Hence, we present this simpler construction in order to highlight our main techniques.

Theorem 5. *Protocol $\pi_{\mathsf{TLP-Light}}$ UC-realizes $\mathcal{F}_{\mathsf{tlp}}$ in the $\mathcal{G}_{\mathsf{Clock}}, \mathcal{G}_{\mathsf{rpoRO}}$, $\mathcal{F}_{\mathsf{DKG}}$, $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$-hybrid model against an active static adversary $\mathcal{A}$ corrupting a majority of parties in $\mathcal{W}$. The parameters of $\mathcal{F}_{\mathsf{tlp}}$ are tag space $\mathcal{TAG} = \mathbb{G}\times\mathbb{G}\times\{0,1\}^{2\tau}$, proof space $\mathbb{G} \times \{0,1\}^\tau$, slack parameter $\epsilon = \frac{\Delta_{\mathtt{lo}}}{\Delta_{\mathtt{hi}}}$ and delay parameter $\Gamma = \Delta_{\mathtt{hi}}$ where $(\Delta_{\mathtt{lo}}, \cdot) \leftarrow \min_{t\in\{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathtt{delays}(t, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,|\mathcal{W}|-1}, |\mathcal{W}| - 1)\}$, $(\cdot, \Delta_{\mathtt{hi}}) \leftarrow \max_{t\in\{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathtt{delays}(t, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,|\mathcal{W}|-1}, |\mathcal{W}| - 1)\}$ and $\mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,|\mathcal{W}|-1}$ are the delay functions of the instances of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta,1}}, \ldots, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta,|\mathcal{W}|-1}}$ where $\mathcal{P}_j \in \mathcal{W}$ acts as receiver.*

The proof can be found in Appendix C. The core tasks of its simulator $\mathcal{S}$ are making sure that: 1) every puzzle generated by $\mathcal{A}$ is created at $\mathcal{F}_{\mathsf{tlp}}$; and 2) every puzzle that is solved by $\mathcal{F}_{\mathsf{tlp}}$ in the ideal world is simulated towards $\mathcal{A}$. The first task is accomplished by $\mathcal{S}$ by extracting the message m and proof π from every puzzle generated by $\mathcal{A}$ and sending it to $\mathcal{F}_{\mathsf{tlp}}$. The second task is achieved by simulating an execution of $\pi_{\mathsf{TLP-Light}}$ for solving TLPs provided by $\mathcal{F}_{\mathsf{tlp}}$ and later using the leakage of m, π from $\mathcal{F}_{\mathsf{tlp}}$ to program the restricted programmable random oracles such that the output of the protocol matches m, π.

Constructing a Random Beacon. Notice that our $\mathcal{F}_{\mathsf{tlp}}$ can be used to instantiate the random beacon construction of [7]. In this construction, parties generate randomness by broadcasting (or posting to a public ledger) a PV-TLP containing a random input. After a majority of parties have provided their PV-TLPs, these PV-TLPs are opened by their owners, who present their random input along with a proof that it was contained in their PV-TLP. In case one of the owners does not follow the protocol, the other parties can solve the unopened PV-TLP to

Protocol 10: $\pi_{\mathsf{TLP-Light}}$

$\pi_{\mathsf{TLP-Light}}$ is parameterized by a cyclic group $\mathbb{G}$ of order q with generator g. $\pi_{\mathsf{TLP-Light}}$ is executed by parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, among which a subset of solvers $\mathcal{W} \subset \mathcal{P}$, interacting with $\mathcal{G}_{\mathsf{Clock}}$, ${\mathcal{G}_{\mathsf{rpoRO}}}^1$ with output in $\mathbb{Z}_{|\mathbb{I}|}$, ${\mathcal{G}_{\mathsf{rpoRO}}}^2$ with output in $\{0,1\}^{2\tau}$, $\mathcal{F}_{\mathsf{DKG}}$ and instances $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i}$ where $\mathcal{P}_i$ is sender and $\mathcal{P}_{i+1}$ is receiver for all $\mathcal{P}_i \in \mathcal{W}$.

Setup: When first activated, all $\mathcal{P}_i \in \mathcal{P}$ send $(\textsc{PubKey}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DKG}}$, receiving pk, and all $\mathcal{P}_i \in \mathcal{W}$ additionally send $(\textsc{SecKey}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{DKG}}$, receiving sk_i.

Create puzzle: On input $(\mathsf{CreatePuzzle}, \mathsf{sid}, m)$, $\mathcal{P}_i$ encrypts m using pk following the steps of Definition 5:

1. Sample $r \xleftarrow{\$} \mathbb{G}, s \xleftarrow{\$} \{0,1\}^{\tau}$ and send $(\textsc{Hash-Query}, r)$ to ${\mathcal{G}_{\mathsf{rpoRO}}}^2$, receiving $(\textsc{Hash-Confirm}, \mathsf{pad})$. Then send $(\textsc{Hash-Query}, m|s)$ to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$, receiving $(\textsc{Hash-Confirm}, \rho)$.
2. Send $(\textsc{IsProgrammed}, m|s)$ (resp. $(\textsc{IsProgrammed}, r)$) to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$ (resp. ${\mathcal{G}_{\mathsf{rpoRO}}}^2$) and abort if either of the responses is $(\textsc{IsProgrammed}, 1)$.
3. Compute $\mathsf{puz} = (c_1 = g^{\rho}, c_2 = r \cdot \mathsf{pk}^{\rho}, c_3 = (m|s) \oplus \mathsf{pad})$.
4. Output $(\mathsf{CreatedPuzzle}, \mathsf{sid}, \mathsf{puz}, \pi = (\mathsf{pk}, r, s))$.

Solve: On input $(\textsc{Solve}, \mathsf{sid}, \mathsf{puz})$, $\mathcal{P}_i$ sends $(\textsc{Solve}, \mathsf{sid}, \mathsf{puz})$ to the first $\mathcal{P}_j \in \mathcal{W}$ (*i.e.* $j = \min\{j \mid \mathcal{P}_j \in \mathcal{W}\}$). Upon receiving $(\textsc{Solved}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ from the last $\mathcal{P}_\ell \in \mathcal{W}$ (*i.e.* $\ell = \max\{\ell \mid \mathcal{P}_\ell \in \mathcal{W}\}$), perform **Public Verification** on puz, m, π and set $m = \perp$ if it does not succeed. Output $(\textsc{Solved}, \mathsf{sid}, \mathsf{puz}, m, \pi)$.

Public Verification: On input $(\textsc{Verify}, \mathsf{sid}, \mathsf{puz} = (c_1, c_2, c_3), m, \pi = (\mathsf{pk}, r, s))$, $\mathcal{P}_i$ executes Steps 2 to 5 of **Create Puzzle** with pk, m, r, s to obtain puz'. If $\mathsf{puz}' = \mathsf{puz}$, $\mathcal{P}_i$ sets $b = 1$, else, it sets $b = 0$, outputting $(\textsc{Verified}, \mathsf{sid}, \mathsf{puz}, m, \pi, b)$.

Tick: Parties in $\mathcal{W}$ proceed as follows and then send $(\mathtt{Update})$ to $\mathcal{G}_{\mathsf{Clock}}$:

Starting Solution: For all $(\textsc{Solve}, \mathsf{sid}, \mathsf{puz} = (c_1, c_2, c_3))$ received in this tick, the first $\mathcal{P}_i \in \mathcal{W}$ proceeds as follows: 1. Send $(\textsc{Read})$ to $\mathcal{G}_{\mathsf{Clock}}$, obtaining $(\textsc{Read}, \nu_1)$; 2. Compute $\hat{c}_2 = c_2 \cdot c_1^{-\mathsf{sk}_i}$; 3. Send $(\textsc{Send}, \mathsf{sid}, (\nu_1, \mathsf{puz}, \hat{c}_2))$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},1}$.

Ongoing Solution: Every party $\mathcal{P}_j \in \mathcal{W} \setminus \mathcal{P}_i$ sends $(\textsc{Rec}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta}}$ where they act as receivers and, for every message $(\textsc{Sent}, \mathsf{sid}, (\nu_1, \mathsf{puz}, \hat{c}_2), \nu)$ received as answer, proceed as follows:

1. Given the current time $\overline{\nu}$ obtained from $\mathcal{G}_{\mathsf{Clock}}$, ν and all the delay functions $\mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,j-1}$ associated to the previous instances of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta}}$, check that $\mathsf{isP}(\nu_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,j-1}, \overline{\nu})$ is true, aborting otherwise.
2. Parse $\mathsf{puz} = (c_1, c_2, c_3)$ and compute $\tilde{c}_2 = \hat{c}_2 \cdot c_1^{-\mathsf{sk}_j}$.
3. If $\mathcal{P}_j$ is not the last party $\mathcal{P}_\ell \in \mathcal{W}$, send $(\textsc{Send}, \mathsf{sid}, (\nu_1, \mathsf{puz}, \tilde{c}_2))$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},j}$.

Delivering Result: The last party $\mathcal{P}_\ell \in \mathcal{W}$ obtains $r = \tilde{c}_2 = c_2 \cdot c_1^{-\sum_{j \in \mathcal{W}} \mathsf{sk}_j}$, sends $(\textsc{Hash-Query}, r)$ to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$, receiving $(\textsc{Hash-Confirm}, \mathsf{pad})$, computes $m|s = c_3 \oplus \mathsf{pad}$ and broadcasts $(m, \pi = (\mathsf{pk}, r, s))$ to all $\mathcal{P}_i \in \mathcal{P}$.

obtain the remaining random input. Finally all parties hash all random inputs to obtain a random output. In our setting, this is particularly advantageous, since potentially sequential communication delay channels only needs to be used in case a party misbehaves. When there is no misbehavior, randomness can be obtained cheaply by locally verifying PV-TLP proofs without accessing delayed channels. Otherwise, if sequential communication delay must be used, a party

who failed to open their PV-TLP is identified, so it can be excluded in future executions and/or made to pay for access to delay channels.

8 Stateless VDF

This Delay Encryption construction from 6 can also be converted into a stateless VDF. Since we combine standard results in order to obtain this construction, we only informally sketch it here. In Section 5 we have described a VDF construction that creates the random value from a proof of sequential delay. Unfortunately, in order to achieve uniqueness we have to use a bulletin board to keep track of previous VDF inputs. Departing from our Delay Encryption construction, obtaining a stateless VDF is possible as follows: assume that a threshold instance of $\texttt{IBE}$ is set up such that $\texttt{Setup}$ was run and pk is known. To evaluate the VDF, consider the VDF input x as an ID and run the threshold version of $\texttt{KG}$ to generate sk_x. Then, hashing x, sk_x using a random oracle yields the VDF output, while sk_x serves as the publicly verifiable proof[6]. Unpredictability follows due to the Naor transform [23], since each sk_x can be considered as a signature of an EUF-CMA secure signature scheme (which is therefore UC secure). Uniqueness of the signature follows from the El Gamal-type of $\texttt{IBE}$, as each sk_x is unique. The VDF delay is then identical with the runtime of $\texttt{KG}$. We formalize this result in the following theorem, which is conservatively phrased in terms of the [12] IBE, although it can be generalized to any IBE that yields a unique signature via the Naor Transform and supports distributed key generation.

Theorem 6. *If the IBE scheme of [12] is* $\mathrm{IND-ID-CCA2}$ *secure, there exists a protocol that UC-realizes* $\mathcal{F}_{\mathsf{VDF}}$ *in the* $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}, \mathcal{F}_{\mathrm{NIZK}}, \mathcal{G}_{\mathsf{rpoRO}}$*-hybrid model against an active static adversary corrupting a majority of parties in* $\mathcal{P}$*. The delay parameter is* $\Gamma = \Delta_{\texttt{hi}}$ *and the slack parameter is* $\epsilon = \frac{\Delta_{\texttt{lo}}}{\Delta_{\texttt{hi}}}$ *where* $(\cdot, \Delta_{\texttt{hi}}) = \max_{t\in\{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$ *and* $(\Delta_{\texttt{lo}}, \cdot) = \min_{t\in\{0,\ldots,\mathsf{poly}(\tau)\}}\{\mathsf{f}_\Delta(t)\}$.

Acknowledgment

The work described in this paper has received funding from: the Protocol Labs Research Grant Program PL-RGP1-2021-064, the Protocol Labs-CryptoSat SpaceVDF program, the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B, and the VR project number 2022-04684.

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained

[6] Which can be checked by encrypting a random value to identity x, decrypting using sk_x and checking for consistency

herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

1. Cryptosat. https://cryptosat.io. Accessed: 2022-10-07.
2. Pouriya Alikhani, Nicolas Brunner, Claude Crépeau, Sébastien Designolle, Raphaël Houlmann, Weixu Shi, Nan Yang, and Hugo Zbinden. Experimental relativistic zero-knowledge proofs. *Nat.*, 599(7883):47–50, 2021.
3. Ghada Almashaqbeh, Ran Canetti, Yaniv Erlich, Jonathan Gershoni, Tal Malkin, Itsik Pe'er, Anna Roitburd-Berman, and Eran Tromer. Unclonable polymers and their cryptographic applications. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 759–789. Springer, Heidelberg, May / June 2022.
4. Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 324–356. Springer, Heidelberg, August 2017.
5. Roger R Bate, Donald D Mueller, Jerry E White, and William W Saylor. *Fundamentals of astrodynamics*. Courier Dover Publications, 2020.
6. Carsten Baum, Bernardo David, and Rafael Dowsley. (Public) Verifiability for Composable Protocols Without Adaptivity or Zero-Knowledge. volume 13600 of *LNCS*, pages 249–272. Springer, 2022.
7. Carsten Baum, Bernardo David, Rafael Dowsley, Ravi Kishore, Jesper Buus Nielsen, and Sabine Oechsner. CRAFT: Composable randomness beacons and output-independent abort MPC from time. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 439–470. Springer, Heidelberg, May 2023.
8. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 429–459. Springer, Heidelberg, October 2021.
9. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.
10. Alexandra Boldyreva, Craig Gentry, Adam O'Neill, and Dae Hyun Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 2007*, pages 276–285. ACM Press, October 2007.
11. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.
12. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

13. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Heidelberg, August 2000.
14. Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70. Springer, Heidelberg, August 2011.
15. Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 302–326. Springer, Heidelberg, October 2021.
16. Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. The wonderful world of global random oracles. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, April / May 2018.
17. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
18. Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219. IEEE Computer Society, 2004.
19. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.
20. Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 380–403. Springer, Heidelberg, March 2006.
21. Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions (extended abstract). In *29th FOCS*, pages 42–52. IEEE Computer Society Press, October 1988.
22. Claude Crépeau, Arnaud Massenet, Louis Salvail, Lucas Shigeru Stinchcombe, and Nan Yang. Practical relativistic zero-knowledge for NP. In Yael Tauman Kalai, Adam D. Smith, and Daniel Wichs, editors, *ITC 2020*, pages 4:1–4:18. Schloss Dagstuhl, June 2020.
23. Yang Cui, Eiichiro Fujisaki, Goichiro Hanaoka, Hideki Imai, and Rui Zhang. Formal security treatments for signatures from identity-based encryption. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, volume 4784 of *LNCS*, pages 218–227. Springer, Heidelberg, November 2007.
24. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277. Springer, Heidelberg, December 2019.
25. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 125–154. Springer, Heidelberg, May 2020.
26. 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, Part III*, volume 13044 of *LNCS*, pages 447–479. Springer, Heidelberg, November 2021.

27. Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *PKC'99*, volume 1560 of *LNCS*, pages 53–68. Springer, Heidelberg, March 1999.
28. Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, February 2010.
29. Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, Heidelberg, May 2007.
30. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, November 2020.
31. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.
32. Adrian Kent. Unconditionally secure bit commitment. *Physical Review Letters*, 83(7):1447, 1999.
33. Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.
34. Tommaso Lunghi, Jedrzej Kaniewski, Felix Bussières, Raphael Houlmann, Marco Tomamichel, Stephanie Wehner, and Hugo Zbinden. Practical relativistic bit commitment. *Physical Review Letters*, 115(3):030502, 2015.
35. Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, and Hovav Shacham. Sequential aggregate signatures from trapdoor permutations. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 74–90. Springer, Heidelberg, May 2004.
36. Ueli M. Maurer. Protocols for secret key agreement by public discussion based on common information. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 461–470. Springer, Heidelberg, August 1993.
37. Ryo Nishimaki, Yoshifumi Manabe, and Tatsuaki Okamoto. Universally composable identity-based encryption. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 2006*, 2006.
38. Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. Physical one-way functions. *Science*, 297(5589):2026–2030, 2002.
39. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.
40. David Pointcheval. Chosen-ciphertext security for any one-way cryptosystem. In Hideki Imai and Yuliang Zheng, editors, *PKC 2000*, volume 1751 of *LNCS*, pages 129–146. Springer, Heidelberg, January 2000.
41. J. Puig-Suari, C. Turner, and W. Ahlgren. Development of the standard cubesat deployer and a cubesat class picosatellite. In *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*, volume 1, pages 1/347–1/353 vol.1, 2001.
42. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto, 1996.
43. Ulrich Rührmair and Marten van Dijk. On the practical use of physical unclonable functions in oblivious transfer and bit commitment protocols. *Journal of Cryptographic Engineering*, 3:17–28, 2013.

44. David A Vallado. *Fundamentals of astrodynamics and applications*, volume 12. Springer Science & Business Media, 2001.
45. Ephanielle Verbanis, Anthony Martin, Raphaël Houlmann, Gianluca Boso, Félix Bussières, and Hugo Zbinden. 24-hour relativistic bit commitment. *Phys. Rev. Lett.*, 117:140506, Sep 2016.
46. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

A Auxiliary Functionalities and other Preliminaries

We use the (Global) Universal Composability or (G)UC model [17, 19] for analyzing security and refer interested readers to the original works for more details.

In UC protocols are run by interactive Turing Machines (iTMs) called *parties*. A protocol π will have n parties which we denote as $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$. The *adversary* $\mathcal{A}$, which is also an iTM, can corrupt a subset $I \subset \mathcal{P}$ as defined by the security model and gains control over these parties. The parties can exchange messages via resources, called *ideal functionalities* (which themselves are iTMs) and which are denoted by $\mathcal{F}$.

As usual, we define security with respect to an iTM $\mathcal{Z}$ called *environment*. The environment provides inputs to and receives outputs from the parties $\mathcal{P}$. To define security, let $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ be the distribution of the output of an arbitrary $\mathcal{Z}$ when interacting with $\mathcal{A}$ in a real protocol instance π using resources $\mathcal{F}_1, \ldots$. Furthermore, let $\mathcal{S}$ denote an *ideal world adversary* and $\mathcal{F} \circ \mathcal{S}$ be the distribution of the output of $\mathcal{Z}$ when interacting with parties which run with $\mathcal{F}$ instead of π and where $\mathcal{S}$ takes care of adversarial behavior.

Definition 2. *We say that $\mathcal{F}$ UC-securely implements π if for every iTM $\mathcal{A}$ there exists an iTM $\mathcal{S}$ (with black-box access to $\mathcal{A}$) such that no environment $\mathcal{Z}$ can distinguish $\pi^{\mathcal{F}_1,\cdots} \circ \mathcal{A}$ from $\mathcal{F} \circ \mathcal{S}$ with non-negligible probability.*

In the security experiment $\mathcal{Z}$ may arbitrarily activate parties or $\mathcal{A}$, though *only one iTM (including $\mathcal{Z}$) is active at each point of time.*

The Global Random Oracle. In Functionality 11 we present the restricted observable and programmable global random oracle ideal functionality from [16]. It follows the standard notion of a random oracle, when defined in the UC framework.

Key Registration Ideal Functionality $\mathcal{F}_{\mathsf{Reg}}$. The key registration functionality $\mathcal{F}_{\mathsf{Reg}}$ is presented in Functionality 12. This ideal functionality captures a public key infrastructure, allowing parties to register their public keys in such a way that other parties can retrieve public keys with the guarantee that they belong to the party who originally registered them. $\mathcal{F}_{\mathsf{Reg}}$ is inspired by the functionality from [20], but additionally supports timestamps on registered keys.

Functionality 11: $\mathcal{G}_{\mathsf{rpoRO}}$

$\mathcal{G}_{\mathsf{rpoRO}}$ is parameterized by an output size function ℓ and a security parameter τ, and keeps initially empty lists $\mathsf{List}_{\mathcal{H}}$,$\mathsf{prog}$.

Query: On input (HASH-QUERY, m) from party $(\mathcal{P}, \mathsf{sid})$ or $\mathcal{S}$, parse m as (s, m') and proceed as follows:

1. Look up h such that $(m, h) \in \mathsf{List}_{\mathcal{H}}$. If no such h exists, sample $h \xleftarrow{\$} \{0,1\}^{\ell(\tau)}$ and set $\mathsf{List}_{\mathcal{H}} = \mathsf{List}_{\mathcal{H}} \cup \{(m, h)\}$.
2. If this query is made by $\mathcal{S}$, or if $s \neq \mathsf{sid}$, then add (s, m', h) to the (initially empty) list of illegitimate queries $\mathcal{Q}_s$.
3. Send (HASH-CONFIRM, h) to the caller.

Observe: On input (OBSERVE, sid) from $\mathcal{S}$, if $\mathcal{Q}_{\mathsf{sid}}$ does not exist yet, set $\mathcal{Q}_{\mathsf{sid}} = \emptyset$. Output (LIST-OBSERVE, $\mathcal{Q}_{\mathsf{sid}}$) to $\mathcal{S}$.

Program: On input (PROGRAM-RO, m, h) with $h \in \{0,1\}^{\ell(\tau)}$ from $\mathcal{S}$, ignore the input if there exists $h' \in \{0,1\}^{\ell(\tau)}$ where $(m, h') \in \mathsf{List}_{\mathcal{H}}$ and $h \neq h'$. Otherwise, set $\mathsf{List}_{\mathcal{H}} = \mathsf{List}_{\mathcal{H}} \cup \{(m, h)\}$, $\mathsf{prog} = \mathsf{prog} \cup \{m\}$ and send (PROGRAM-CONFIRM) to $\mathcal{S}$.

IsProgrammed: On input (ISPROGRAMMED, m) from a party $\mathcal{P}$ or $\mathcal{S}$, if the input was given by $(\mathcal{P}, \mathsf{sid})$ then parse m as (s, m') and, if $s \neq \mathsf{sid}$, ignore this input. Set $b = 1$ if $m \in \mathsf{prog}$ and $b = 0$ otherwise. Then send (ISPROGRAMMED, b) to the caller.

Functionality 12: $\mathcal{F}_{\mathsf{Reg}}$

$\mathcal{F}_{\mathsf{Reg}}$ interacts with a set of parties $\mathcal{P}$ and an ideal adversary $\mathcal{S}$ as well as a global clock $\mathcal{G}_{\mathsf{Clock}}$ as follows:

Key Registration: Upon receiving a message (REGISTER, $\mathsf{sid}, \mathsf{pk}$) from a party $\mathcal{P}_i \in \mathcal{P}$:

1. Send (READ) to $\mathcal{G}_{\mathsf{Clock}}$, waiting for response (READ, ν).
2. Send (REGISTERING, $\mathsf{sid}, \mathsf{pk}, \mathcal{P}_i, \nu$) to $\mathcal{S}$. Upon receiving $(\mathsf{sid}, ok, \mathcal{P}_i)$ from $\mathcal{S}$, and if this is the first message from $\mathcal{P}_i$, then record the tuple $(\mathcal{P}_i, \mathsf{pk}, \nu)$.

Key Retrieval: Upon receiving a message (RETRIEVE, $\mathsf{sid}, \mathcal{P}_j$) from a party $\mathcal{P}_i \in \mathcal{P}$, send message (RETRIEVE, $\mathsf{sid}, \mathcal{P}_j$) to $\mathcal{S}$ and wait for it to return a message (RETRIEVE, sid, ok). Then, if there is a recorded tuple $(\mathcal{P}_j, \mathsf{pk}, \nu)$ output (RETRIEVE, $\mathsf{sid}, \mathcal{P}_j, \mathsf{pk}, \nu$) to $\mathcal{P}_i$. Otherwise, if there is no recorded tuple, return (RETRIEVE, $\mathsf{sid}, \mathcal{P}_j, \perp$).

Unique Digital Signatures Ideal Functionality $\mathcal{F}_{\mathsf{Sig}}$. The standard digital signature functionality $\mathcal{F}_{\mathsf{Sig}}$ from [18] captures a randomized signature scheme where the signer may influence the generation of a signature by choosing the randomness used by the signing algorithm. This particularity is captured by allowing the ideal adversary $\mathcal{S}$ choose a new string σ to represent a signature on a message m every time the signer $\mathcal{P}_s$ (a special party who has the right to generate signatures, *i.e.*, who holds the signature key) makes a new request for a signature on m. This process allows for multiple valid signatures to be produced for the same message. However, we require a unique signature scheme for our applications to

Functionality 13: $\mathcal{F}_{\mathsf{Sig}}$

Given an ideal adversary $\mathcal{S}$, verifiers $\mathcal{V}$ and a signer $\mathcal{P}_s$, $\mathcal{F}_{\mathsf{Sig}}$ performs:

Key Generation: Upon receiving a message $(\text{KEYGEN}, \mathsf{sid})$ from $\mathcal{P}_s$, verify that $\mathsf{sid} = (\mathcal{P}_s, \mathsf{sid}')$ for some sid'. If not, ignore the request. Else, hand $(\text{KEYGEN}, \mathsf{sid})$ to the adversary $\mathcal{S}$. Upon receiving $(\text{VERIFICATION KEY}, \mathsf{sid}, \mathsf{SIG}.vk)$ from $\mathcal{S}$, output $(\text{VERIFICATION KEY}, \mathsf{sid}, \mathsf{SIG}.vk)$ to $\mathcal{P}_s$, and record the pair $(\mathcal{P}_s, \mathsf{SIG}.vk)$.

Signature Generation: Upon receiving a message $(\text{SIGN}, \mathsf{sid}, m)$ from $\mathcal{P}_s$, verify that $\mathsf{sid} = (\mathcal{P}_s, \mathsf{sid}')$ for some sid' . If not, then ignore the request. Else, if an entry $(m, \sigma, \mathsf{SIG}.vk, 1)$ is recorded, output $(\text{SIGNATURE}, \mathsf{sid}, m, \sigma)$ to $\mathcal{P}_s$ and ignore the next steps (this condition guarantees uniqueness). Else, send $(\text{SIGN}, \mathsf{sid}, m)$ to $\mathcal{S}$. Upon receiving $(\text{SIGNATURE}, \mathsf{sid}, m, \sigma)$ from $\mathcal{S}$, verify that no entry $(m, \sigma, \mathsf{SIG}.vk, 0)$ is recorded. If it is, then output an error message to $\mathcal{P}_s$ and halt. Else, output $(\text{SIGNATURE}, \mathsf{sid}, m, \sigma)$ to $\mathcal{P}_s$, and record the entry $(m, \sigma, \mathsf{SIG}.vk, 1)$.

Signature Verification: Upon receiving a message $(\text{VERIFY}, \mathsf{sid}, m, \sigma, \mathsf{SIG}.vk')$ from some party $\mathcal{V}_i \in \mathcal{V}$, hand $(\text{VERIFY}, \mathsf{sid}, m, \sigma, \mathsf{SIG}.vk')$ to $\mathcal{S}$. Upon receiving $(\text{VERIFIED}, \mathsf{sid}, m, \phi)$ from $\mathcal{S}$ do:

1. If $\mathsf{SIG}.vk' = \mathsf{SIG}.vk$ and the entry $(m, \sigma, \mathsf{SIG}.vk, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key $\mathsf{SIG}.vk'$ is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
2. Else, if $\mathsf{SIG}.vk' = \mathsf{SIG}.vk$, the signer $\mathcal{P}_s$ is not corrupted, and no entry $(m, \sigma', \mathsf{SIG}.vk, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, \mathsf{SIG}.vk, 0)$. (This condition guarantees unforgeability: If $\mathsf{SIG}.vk'$ is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
3. Else, if there is an entry $(m, \sigma, \mathsf{SIG}.vk', f')$ recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $f = \phi$ and record the entry $(m, \sigma, \mathsf{SIG}.vk', \phi)$.

Output $(\text{VERIFIED}, \mathsf{sid}, m, f)$ to $\mathcal{V}_i$.

proofs of sequential communication. In a unique signature scheme, only one signature may be produced for a given message m under a signing key. In the UC formalization of signature schemes, an instance of the functionality $\mathcal{F}_{\mathsf{Sig}}$ itself represents each different signing key by allowing only a special party $\mathcal{P}_s$ (*i.e.* the holder of a signing key) to produce signatures. Hence, we capture the notion of unique signatures by only allowing one signature on a given message m to be produced by the same instance of $\mathcal{F}_{\mathsf{Sig}}$. The remainder of this functionality still follows the same steps as the standard one from [18]. Our modified $\mathcal{F}_{\mathsf{Sig}}$ capturing unique signatures is presented in Functionality 13, where modifications with respect to [18] are written in this font.

It is shown in [18] that any EUF-CMA signature scheme UC realizes the standard signature functionality where multiple valid signatures may be pro-

duced for the same message under the same signing key (*i.e.* the same instance of $\mathcal{F}_{\mathsf{Sig}}$ may generate multiple signatures for the same message, as long as they have not been flagged as invalid signatures by a previous unsuccessful verification procedure). We observe that this fact trivially extends to the case of unique signatures, *i.e.*, any EUF-CMA signature scheme UC realizes our $\mathcal{F}_{\mathsf{Sig}}$ capturing unique signatures, since the only restriction in this case is that a single signature is produced for each message by a single instance of $\mathcal{F}_{\mathsf{Sig}}$ (which represents a signer's signing key).

Bulletin Board Ideal Functionality $\mathcal{F}_{\mathsf{BB}}$. In Functionality 14 we describe an authenticated bulletin board functionality which is used throughout this work. Authenticated Bulletin Boards can be constructed from regular bulletin boards using $\mathcal{F}_{\mathsf{Sig}}, \mathcal{F}_{\mathsf{Reg}}$ and standard techniques.

Functionality 14: $\mathcal{F}_{\mathsf{BB}}$

$\mathcal{F}_{\mathsf{BB}}$ interacts with a set of parties $\mathcal{P}$ and keeps a counter c initially set to 0, proceeding as follows:

Write: Upon receiving $(\textsc{Write}, \mathsf{sid}, m)$ from $\mathcal{P}_i \in \mathcal{P}$, store the message (c, m) and increment c.

Read: Upon receiving $(\textsc{Read}, \mathsf{sid})$ from $\mathcal{P}_i \in \mathcal{P}$, return all messages $(\cdot, m)$ that are stored.

A.1 UC Secure Public-Key Encryption with Plaintext Verification

Semantics of a public-key encryption scheme. We consider public-key encryption schemes PKE that have public-key $\mathcal{PK}$, secret key $\mathcal{SK}$, message $\mathcal{M}$, randomness $\mathcal{R}$ and ciphertext $\mathcal{C}$ spaces that are functions of the security parameter τ, and consist of a PPT key generation algorithm KG, a PPT encryption algorithm Enc and a deterministic decryption algorithm Dec. For $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KG}(1^\tau)$, any $\mathsf{m} \in \mathcal{M}$, and $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, it should hold that $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = \mathsf{m}$ with overwhelming probability over the used randomness.

Moreover, we extend the semantics of public-key encryption by adding a plaintext verification algorithm $\{0,1\} \leftarrow \mathsf{V}(\mathsf{ct}, \mathsf{m}, \pi)$ that outputs 1 if m is the plaintext message contained in ciphertext ct given a valid proof π that also contains the public-key pk used to generate the ciphertext. Furthermore, we modify the encryption and decryption algorithms as follows: $(\mathsf{ct}, \pi) \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$ and $(\mathsf{m}, \pi) \leftarrow \mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ now output a valid proof π that m is contained in ct. The security guarantees provided by the verification algorithm are laid out in Definition 3.

Definition 3 (Plaintext Verification). *Let $\mathsf{PKE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{V})$ be a public-key encryption scheme and τ be a security parameter. Then PKE has plaintext verification if for every PPT adversary $\mathcal{A}$, it holds that:*

$$\Pr\left[\mathsf{V}(\mathsf{ct},\mathsf{m}',\pi')=1 \;\middle|\; \begin{array}{c}\mathsf{pk}\xleftarrow{\$}\mathcal{PK},(\mathsf{m},\pi,\mathsf{m}',\pi')\xleftarrow{\$}\mathcal{A}(\mathsf{pk}),\\ \pi=(\mathsf{pk},r),\pi'=(\mathsf{pk},r')\in\mathcal{PK}\cup\mathcal{R},\\ \mathsf{m},\mathsf{m}'\in\mathcal{M},(\mathsf{ct},\pi)\leftarrow\mathsf{Enc}(\mathsf{pk},\mathsf{m};r),\mathsf{m}'\neq\mathsf{m}\end{array}\right]\in\mathsf{negl}(\tau)$$

IND-CCA secure Cryptosystem with Plaintext Verification based on [40] from [6]. This cryptosystem can be constructed from any Partially Trapdoor One-Way Injective Function in the random oracle model. Moreover, as observed in [6], it can be instantiated in the restricted observable and programmable global random oracle model of [16]. First we recall the definition of Partially Trapdoor One-Way Functions.

Definition 4 (Partially Trapdoor One-Way Function [40]). *The function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is said to be partially trapdoor one-way if:*

- *For any given $z = f(x, y)$, it is computationally impossible to get back a compatible x. Such an x is called a partial preimage of z. More formally, for any polynomial time adversary A, its success, defined by $\mathtt{Succ}_{\mathtt{A}} = Pr_{x,y}[\exists y', f(x', y') = f(x, y)|x' = A(f(x, y))]$, is negligible. It is one-way even for just finding partial-preimage, thus partial one-wayness.*
- *Using some extra information (the trapdoor), for any given $z \in f(\mathcal{X} \times \mathcal{Y})$, it is easily possible to get back an x, such that there exists a y which satisfies $f(x, y) = z$. The trapdoor does not allow a total inversion, but just a partial one and it is thus called a partial trapdoor.*

As observed in [40], the classical El Gamal cryptosystem is a partially trapdoor one-way injective function under the Computational Diffie Hellman (CDH) assumption, implying an instantiation of this cryptosystem under CDH. We will later exploit this fact to apply this transformation to a simple threshold version of El Gamal where the encryption procedure and the public key are exactly the same as in the standard scheme, allowing for the construction below to be instantiated. We now recall this generic construction.

Definition 5 (Pointcheval [40] IND-CCA Secure Cryptosystem with Plaintext Verification). *Let $\mathcal{TD}$ be a family of partially trapdoor one-way injective functions and let $H : \{0,1\}^{|m|+\tau} \to \mathcal{Y}$ and $G : \mathcal{X} \to \{0,1\}^{|m|+\tau}$ be random oracles, where $|m|$ is message length. This cryptosystem consists of the algorithms $\mathsf{PKE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}\,\mathsf{V})$ that work as follows:*

- $\mathsf{KG}(1^\tau)$*: Sample a random partially trapdoor one-way injective function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ from $\mathcal{TD}$ and denote its inverse parameterized by the trapdoor by $f^{-1} : \mathcal{Z} \to \mathcal{X}$. The public-key is $\mathsf{pk} = f$ and the secret key is $\mathsf{sk} = (f, f^{-1})$.*
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$*: Sample $r \xleftarrow{\$} \mathcal{X}$ and $s \xleftarrow{\$} \{0,1\}^\tau$. Compute $a \leftarrow f(r, H(\mathsf{m}|s))$ and $b = (\mathsf{m}|s) \oplus G(r)$, outputting $\mathsf{ct} = (a, b)$ as the ciphertext and $\pi = (\mathsf{pk}, r, s)$ as the proof.*

– $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$*: Given a ciphertext* $\mathsf{ct} = (a, b)$ *and secret key* $\mathsf{sk} = f^{-1}$*, compute* $r \leftarrow f^{-1}(a)$ *and* $M \leftarrow b \oplus G(r)$*. If* $a = f(r, H(M))$*, parse* $M = (m|s)$ *and output* m *and the proof* $\pi = (\mathsf{pk}, r, s)$ *. Otherwise, output* $\perp$*.*
– $\mathsf{V}(\mathsf{ct}, \mathsf{m}, \pi)$*: Parse* $\pi = (\mathsf{pk}, r, s)$*, compute* $\mathsf{ct}' \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{m}, (r, s))$ *and output* 1 *if and only if* $\mathsf{ct} = \mathsf{ct}'$*.*

A.2 Global Clocks and Global tickers

We now discuss the [8] model, which expresses time within the Generalized Universal Composability (GUC) framework in such a way that protocols can be made oblivious to clock ticks. Specifically, TARDIS models the passage of time without implying synchronicity. Our results can be stated in this model as well, which makes our results directly comparable and compatible with previous work on UC PV-TLPs and VDFs [7, 8] that adopt the same model.

Global Tickers: In [7, 8], a global ticker functionality $\mathcal{G}_{\mathsf{ticker}}$ (see Functionality 15) keeps track of "ticks" representing a discrete unit of time. When activated by another ideal functionality, the global ticker answers whether or not a new "tick" has happened since the last time it was activated by this ideal functionality but does not provide a synchronized clock value. To ensure that all honest parties can observe all relevant timing-related events, $\mathcal{G}_{\mathsf{ticker}}$ only progresses if all honest parties have signaled that they have been activated (in arbitrary order). Parties do not get outputs from $\mathcal{G}_{\mathsf{ticker}}$. Ticked functionalities can freely interpret ticks and perform arbitrary internal state changes. Upon each activation, any ticked ideal functionality first checks with $\mathcal{G}_{\mathsf{ticker}}$ if a new tick has happened and if yes, executes code in a special Tick interface. In a protocol realizing a ticked functionality, parties activate the global ticker after executing their steps, so that a new tick is allowed to happen. We refer to [8] for more details

Functionality 15: $\mathcal{G}_{\mathsf{ticker}}$

Initialize a set of registered parties $\mathtt{Pa} = \emptyset$, a set of registered functionalities $\mathtt{Fu} = \emptyset$, a set of activated parties $L_{\mathtt{Pa}} = \emptyset$, and a set of functionalities $L_{\mathtt{Fu}} = \emptyset$ that have been informed about the current tick.

Party registration: Upon receiving $(\mathtt{register}, \mathsf{pid})$ from honest party $\mathcal{P}$ with pid pid, add pid to $\mathtt{Pa}$ and send $(\mathtt{registered})$ to $\mathcal{P}$.

Functionality registration: Upon receiving $(\mathtt{register})$ from functionality $\mathcal{F}$, add $\mathcal{F}$ to $\mathtt{Fu}$ and send $(\mathtt{registered})$ to $\mathcal{F}$.

Tick: Upon receiving $(\mathtt{tick})$ from the environment, do the following:
1. If $\mathtt{Pa} = L_{\mathtt{Pa}}$, reset $L_{\mathtt{Pa}} = \emptyset$ and $L_{\mathtt{Fu}} = \emptyset$, and send $(\mathtt{ticked})$ to the adversary $\mathcal{S}$.
2. Else, send $(\mathtt{notticked})$ to the environment.

Ticked request: Upon receiving $(\mathtt{ticked}?)$ from functionality $\mathcal{F} \in \mathtt{Fu}$: If $\mathcal{F} \notin L_{\mathtt{Fu}}$, add $\mathcal{F}$ to $L_{\mathtt{Fu}}$ and send $(\mathtt{ticked})$ to $\mathcal{F}$. Otherwise send $(\mathtt{notticked})$ to $\mathcal{F}$.

Record party activation: Upon receiving $(\mathtt{activated})$ from party $\mathcal{P}$ with pid $\mathsf{pid} \in \mathtt{Pa}$, add pid to $L_{\mathtt{Pa}}$ and send $(\mathtt{recorded})$ to $\mathcal{P}$.

Synchronicity and Global Clocks As mentioned in Section 2 we need to assume that honest parties have synchronized clocks. This is necessary to argue about communication delays that depend on the relative position of two parties, which evolves in time. We capture this notion of synchronicity by using a global clock functionality $\mathcal{G}_{\mathsf{Clock}}$ (see Functionality 1). In the definition that we use throughout the main body, $\mathcal{G}_{\mathsf{Clock}}$ allows users to query the current time and increments an internal time counter once all functionalities and honest parties activate a clock update interface after the last update. However, one can realize $\mathcal{G}_{\mathsf{Clock}}$ in the TARDIS abstract composable time model. To do so, we update its internal time counter when $\mathcal{G}_{\mathsf{ticker}}$ issues a new tick.

Global Clocks in the TARDIS [8] model: In order to integrate the global functionality $\mathcal{G}_{\mathsf{Clock}}$ into the abstract composable time model, we modify it as outlined above. This modification captures the fact that $\mathcal{G}_{\mathsf{Clock}}$ exposes towards the parties and other ideal functionalities the number of ticks issues by $\mathcal{G}_{\mathsf{ticker}}$ since the beginning of the execution. However, it is not a separate clock that is executed independently from $\mathcal{G}_{\mathsf{ticker}}$. Since we wish $\mathcal{G}_{\mathsf{Clock}}$ to count the ticks issued by $\mathcal{G}_{\mathsf{ticker}}$, our modified version of $\mathcal{G}_{\mathsf{Clock}}$ requires all honest parties to activate the global ticker every time they would update the global clock (*i.e.* when they have executed all their instructions for a given round). This modification can be seen in Functionality 16. It is immediate how this clock functionality can be used to replace the global $\mathcal{G}_{\mathsf{Clock}}$ throughout our protocols by replacing UPDATE messages to $\mathcal{G}_{\mathsf{Clock}}$ by ACTIVATED calls to $\mathcal{G}_{\mathsf{ticker}}$.

Functionality 16: $\mathcal{G}_{\mathsf{Clock}}$

$\mathcal{G}_{\mathsf{Clock}}$ interacts with a sets $\mathcal{P}, \mathcal{F}$ of parties and functionalities, respectively, as well as with $\mathcal{G}_{\mathsf{ticker}}$. It keeps a counter ν initially set to 0.

Clock Read: Upon receiving (READ) from any entity, answer with (READ, ν).

Tick: Increment ν, *i.e.* set $\nu \leftarrow \nu + 1$.

B Delayed Communication - Proofs and more details

B.1 Realizing $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$

The multiple-use ideal functionality $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ for authenticated delayed message transmission can be realized in the $\mathcal{G}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}}$-hybrid model. Assume access to many instances of the single-use functionality $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}}$, one fresh instance of $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t}$ associated to t for each message to be sent at time $t \in \{0, \ldots, \mathsf{poly}(\tau)\}$ with parameters $(\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t) \leftarrow \mathsf{f}_\Delta(t)$. Upon receiving an input $(\text{SEND}, \mathsf{sid}, m)$, a sender $\mathcal{P}_S$ determines $(\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t) \leftarrow \mathsf{f}_\Delta(t)$ and uses the instance of $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t}$ to send (m, t). Upon receiving input $(\text{REC}, \mathsf{sid})$, a receiver $\mathcal{P}_R$ queries all instances

of $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^{t'},\Delta_{\mathtt{hi}}^{t'}}$ associated to a time t' smaller than current time t in order to retrieve messages that might have been sent. It then has to establish correctness of the delay.

Protocol 17: π_{mdmt}

For each $t \in \{0,\ldots,\mathsf{poly}(\tau)\}$ let $(\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t) \leftarrow \mathsf{f}_\Delta(t)$. In the protocol two parties $\mathcal{P}_S, \mathcal{P}_R$ interact via functionalities $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^t,\Delta_{\mathtt{hi}}^t}$. In addition, they use a global clock $\mathcal{G}_{\mathsf{Clock}}$. Upon any activation that is not related to a message below, parties send (`Update`) to $\mathcal{G}_{\mathsf{Clock}}$.

Send: Upon input $(\textsc{Send}, \mathsf{sid}, m)$ $\mathcal{P}_S$ acts as follows:
1. Send $(\textsc{Read})$ to $\mathcal{G}_{\mathsf{Clock}}$ and obtain $(\textsc{Read}, \bar{t})$.
2. Determine $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(\bar{t})$.
3. Send $(\textsc{Send}, \mathsf{sid}, (m, \bar{t}))$ to $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ and $(\textsc{Update})$ to $\mathcal{G}_{\mathsf{Clock}}$.

Receive: Upon input $(\textsc{Rec}, \mathsf{sid})$ $\mathcal{P}_R$ acts as follows:
1. Send $(\textsc{Read})$ to $\mathcal{G}_{\mathsf{Clock}}$ and obtain $(\textsc{Read}, \bar{t})$.
2. For each $t \in \{0,\ldots,\bar{t}\}$ compute $(\Delta_{\mathtt{lo}}^t, \Delta_{\mathtt{hi}}^t) \leftarrow \mathsf{f}_\Delta(t)$.
3. Send $(\textsc{Rec}, \mathsf{sid})$ to each $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^t,\Delta_{\mathtt{hi}}^t}$ and wait for responses $(\textsc{Sent}, \mathsf{sid}, (m, t'))$ from $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}}^t,\Delta_{\mathtt{hi}}^t}$. If $t \neq t'$ then $\mathcal{P}_R$ ignores (m, t').
4. If $\Delta_{\mathtt{lo}} + t \leq \bar{t} \leq \Delta_{\mathtt{hi}} + t$ then $\mathcal{P}_R$ outputs $(\textsc{Sent}, \mathsf{sid}, m, t)$.

Theorem 7. *The protocol π_{mdmt} in Protocol 17 GUC-securely implements $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ in the $\mathcal{G}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{dmt}}^{\cdot}$-hybrid model against a static active adversary.*

Proof. We now construct a PPT simulator $\mathcal{S}$ for a corrupted sender or receiver. In both cases, the simulator will simulate all hybrid instances of $\mathcal{F}_{\mathsf{dmt}}^{\cdot}$, which can be done in time polynomial in τ as there are only $\mathsf{poly}(\tau)$ such instances.

If $\mathcal{P}_S$ is corrupted then we construct $\mathcal{S}$ as follows: $\mathcal{S}$ acts like an honest $\mathcal{P}_R$, but it additionally observes all inputs $(\textsc{Send}, \mathsf{sid}, m)$ to any instance of $\mathcal{F}_{\mathsf{dmt}}^{\cdot}$ that it simulates. Any input of the form (m, t) to $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ with $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) = \mathsf{f}_\Delta(t)$ is forwarded as $(\textsc{Send}, \mathsf{sid}, m)$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ during the same tick of $\mathcal{G}_{\mathsf{Clock}}$. When the adversary makes this $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ output the message (m, t'), then $\mathcal{S}$ makes $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ output m in the same tick round of $\mathcal{G}_{\mathsf{Clock}}$ by sending $(\textsc{ok}, \mathsf{sid}, t')$. This simulation is perfect, as $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ will output any message in the same round where the respective instance of $\mathcal{F}_{\mathsf{dmt}}^{\cdot}$ would have released it to an honest receiver. Moreover, only those messages are forwarded by $\mathcal{S}$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ that wouldn't be ignored by an honest receiver.

If $\mathcal{P}_R$ is corrupted then $\mathcal{S}$ sends $(\textsc{Rec}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ in every tick round. Upon obtaining $(\textsc{Sent}, \mathsf{sid}, m, t')$ from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ in tick round t, $\mathcal{S}$ computes $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t')$ and programs the respective instance $\mathcal{F}_{\mathsf{dmt}}^{\Delta_{\mathtt{lo}},\Delta_{\mathtt{hi}}}$ to contain the message (m, t') and have $\mathtt{msg} = \mathtt{released} = \top$ so that the honest receiver can pick up the message. Again, the simulation is perfect because the instance that is reprogrammed by $\mathcal{S}$ is the one an honest sender would provide the respective

input to. Moreover, given the construction of $\mathcal{F}_{\mathsf{dmt}}$ the dishonest receiver would not be able to obtain the message any earlier than in this round in the real protocol. □

B.2 Proof of Theorem 1

Proof. We construct a PPT simulator $\mathcal{S}$ that emulates the protocol interaction for a corrupted $\mathcal{P}_S$ or $\mathcal{P}_R$. $\mathcal{S}$ will simulate the instances of $\mathcal{F}_{\mathsf{Sig}}, \mathcal{F}_{\mathsf{Reg}}, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$. During **Setup**, $\mathcal{S}$ will in either case of corruption act like an honest party, setting up both instances ${\mathcal{F}_{\mathsf{Sig}}}^S, {\mathcal{F}_{\mathsf{Sig}}}^R$ and will simulate posting its key on $\mathcal{F}_{\mathsf{Reg}}$.

If $\mathcal{P}_S$ is corrupted, then $\mathcal{S}$ during **Send** extracts the message m from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ and checks that $\mathcal{P}_S$ has a key $\mathsf{SIG}.vk_S$ registered with $\mathcal{F}_{\mathsf{Reg}}$ before the current tick round. If the signature verifies with ${\mathcal{F}_{\mathsf{Sig}}}^S$, then forward it to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ as (SEND, sid, m) in the same tick round. When the adversary makes $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ output the message and if an honest verifier would have accepted it, then $\mathcal{S}$ computes $\pi_{\mathtt{lo}}$ as in the protocol using ${\mathcal{F}_{\mathsf{Sig}}}^R$ for its signature. Finally, it lets $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ deliver the message to the honest receiver and sends (PROOF, $\mathsf{sid}, m, t, \pi_{\mathtt{lo}}$) to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. If the timestamp in $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ does not coincide with when the message was sent, it instead lets $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ deliver the message and sends (NOPROOF, sid). For any message (VERIFY, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}$) where $\mathcal{S}$ did generate $\pi_{\mathtt{lo}}$ for this m, t and $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_\Delta(t), \Delta \in [\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}]$ send (VERIFY, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, 1$), otherwise send (VERIFY, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, 0$) to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$.

If instead $\mathcal{P}_R$ is corrupted, wait until $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ outputs (SENT, sid, m, t), then create a valid signature σ_S using $\mathcal{F}_{\mathsf{Sig}}$ and make $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ output (SENT, $\mathsf{sid}, (m, t, \sigma_S), t$) to $\mathcal{P}_R$ in the same tick round. In addition, send (NOPROOF, sid) to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. Then, upon query (VERIFY, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}$) from $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ and if $\pi_{\mathtt{lo}}$ can be parsed as (σ'_S, σ_R), check that $\sigma_S = \sigma'_S$. If not, then send (VERIFIED, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, 0$) to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$. Otherwise emulate the call (VERIFY, $\mathsf{sid}, (m, t, \sigma_S), \sigma_R, \mathsf{SIG}.vk_R$) on ${\mathcal{F}_{\mathsf{Sig}}}^R$ with the adversary, which will ultimately output (VERIFIED, $\mathsf{sid}, (m, t, \sigma_S), f$) to $\mathcal{S}$. Send (VERIFIED, $\mathsf{sid}, m, t, \Delta, \pi_{\mathtt{lo}}, f$) to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$.

Clearly, the simulation runs in polynomial time. For a corrupted $\mathcal{P}_S$, we only make $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ output a proof (and let it later verify a proof positively) if the message from $\mathcal{P}_S$ via $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_\Delta}$ was well-formed. This is identical to the protocol, and also the proof $\pi_{\mathtt{lo}}$ is identical. For the corrupt $\mathcal{P}_R$ we make the simulated protocol output the correctly signed message to it in the same round as it would in the real protocol. Moreover, $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$'s **Verify** responses are consistent with the outputs from the protocol by letting $\mathcal{S}$ verify signatures with $\mathcal{F}_{\mathsf{Sig}}$ first. Hence both cases are perfectly indistinguishable. □

B.3 Computing channel delays

We now define the algorithm $\mathsf{delays}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}, k)$ that works for any threshold $k < n$ of corrupted parties to determine the minimal and maximal observable delay as follows:

1. For $i \in [n-1]$ let $\Delta_{\mathtt{hi}}^i = \max_{j \in \mathsf{poly}(\tau)}\{\Delta_{\mathtt{hi}} \mid (\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}}) \leftarrow \mathsf{f}_{\Delta,i}(j)\}$. Then

$$\Delta_{\mathtt{hi}} = \max_{j \in [\Delta_{\mathtt{hi}}^1 + \cdots + \Delta_{\mathtt{hi}}^{n-1}]} \{j \mid \mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_1 + j)\}$$

2. First party honest:

$$a_1 = \min_{t_1 \le t \le t_1 + \Delta_{\mathtt{hi}}} \{t - t_1 \mid \mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-k}, t_1 + t)\}$$

3. Last party honest:

$$a_2 = \min_{t_1 \le t < t_n \le t_1 + \Delta_{\mathtt{hi}}} \left\{ t_n - t \;\middle|\; \begin{array}{c} \mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,k}, t) \wedge \\ \mathsf{isP}(t, \mathsf{f}_{\Delta,k+1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n) \end{array} \right\}$$

4. First and last two corrupt:

$$a_3 = \min_{i \in \{2,\ldots,k-2\}, t_1 \le t < t' \le t_1 + \Delta_{\mathtt{hi}}} \left\{ t' - t \;\middle|\; \begin{array}{c} \mathsf{isP}(t_1, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,i}, t) \wedge \\ \mathsf{isP}(t, \mathsf{f}_{\Delta,i+1}, \ldots, \mathsf{f}_{\Delta,i+n-k}, t') \wedge \\ \mathsf{isP}(t', \mathsf{f}_{\Delta,i+n-k+1}, \ldots, \mathsf{f}_{\Delta,n-1}, t_n) \end{array} \right\}$$

5. Set $\Delta_{\mathtt{lo}} = \min\{a_1, a_2, a_3\}$ and output $(\Delta_{\mathtt{lo}}, \Delta_{\mathtt{hi}})$.

Clearly, each step of delays makes only polynomially many calls to isP, so the algorithm remains efficient for $n = \mathsf{poly}(\log \tau)$.

Proposition 3. *The algorithm* delays *computes the minimal and maximal observable delay for* k *corruptions of* n *parties given delay functions* $\mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-1}$.

Proof. Clearly, $\Delta_{\mathtt{hi}}$ cannot be larger than the sum of the largest individual delays that any $\mathsf{f}_{\Delta,i}$ can contribute. Hence, $\Delta_{\mathtt{hi}}$ as computed is the largest achievable delay in any observable protocol.

a_1 considers the case where the first $n-k$ parties are honest. That the given statement finds the smallest possible delay in this case follows directly.

Step a_2 considers the case where the last $n-k$ parties are honest. Here, since $\mathcal{P}_{k+1}$ can observe the behavior of $\mathcal{P}_k$ (which is dishonest), the minimal delay includes the delay from $\mathcal{P}_k$ to $\mathcal{P}_{k+1}$.

Finally, step a_3 considers all cases where there are two parties in the beginning and the end of the chain that are corrupted, and picks the best way of having i corrupted in the beginning and $k-i$ in the end so that the honest parties have minimal observable delay. Then, the minimal of all these 3 mutually exclusive cases yields the minimal channel delay. □

B.4 The protocol $\pi_{\mathsf{Multi-SCD}}$ and proof of Theorem 2

Proof. We construct a simulator $\mathcal{S}$ that works for every set of corrupted parties. Let $\mathcal{P}_i$ be the first honest party and $\mathcal{P}_j$ be the last honest party (where $\mathcal{P}_i = \mathcal{P}_j$ is possible). In general, $\mathcal{S}$ will run a simulation of $\pi_{\mathsf{Multi-SCD}}$ with the adversary where it lets every uncorrupted $\mathcal{P}_i$ act honestly, subject to the modifications outlined below.

Protocol 18: $\pi_{\mathsf{Multi-SCD}}$

This protocol is executed by a sender $\mathcal{P}_1$, a set of intermediate parties $\mathcal{P}_2, \ldots, \mathcal{P}_{n-1}$ and a receiver $\mathcal{P}_n$, as well as a set of verifiers $\mathcal{V}$ interacting with each other and with $\mathcal{G}_{\mathsf{Clock}}, \mathcal{F}_{\mathsf{Reg}}, {\mathcal{F}_{\mathsf{Sig}}}^1, \ldots, {\mathcal{F}_{\mathsf{Sig}}}^n$. Each pair $\mathcal{P}_i, \mathcal{P}_{i+1}$ is connected by $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i}$.
In every step, the activated party sends (READ) to $\mathcal{G}_{\mathsf{Clock}}$ to obtain $(\text{READ}, \bar{t})$.

Setup: Upon first activation, each $\mathcal{P}_i$ proceeds as follows:

1. Send $(\text{KEYGEN}, \mathsf{sid})$ to ${\mathcal{F}_{\mathsf{Sig}}}^i$ where $\mathcal{P}_i$ acts as signer.
2. Upon receiving $(\text{VERIFICATION KEY}, \mathsf{sid}, \mathsf{SIG}.vk_i)$ from ${\mathcal{F}_{\mathsf{Sig}}}^i$, $\mathcal{P}_i$ sends $(\text{REGISTER}, \mathsf{sid}, \mathsf{SIG}.vk_i)$ to $\mathcal{F}_{\mathsf{Reg}}$.

Send: Upon receiving first input $(\text{SEND}, \mathsf{sid}, m)$ for $\bar{t}$, $\mathcal{P}_1$ proceeds as follows:

1. Send $(\text{SIGN}, \mathsf{sid}, (m, \bar{t}))$ to ${\mathcal{F}_{\mathsf{Sig}}}^1$, receiving $(\text{SIGNATURE}, \mathsf{sid}, (m, \bar{t}), \sigma_1)$.
2. Send $(\text{SEND}, \mathsf{sid}, (m, \bar{t}, \sigma_1))$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},1}$.

Receive: Upon receiving $(\text{REC}, \mathsf{sid})$, $\mathcal{P}_n$ sends $(\text{REC}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},n-1}$ and proceeds as follows for the first $(\text{SENT}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{n-1}), t')$ received from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},n-1}$:

1. Check if $\mathsf{isP}(t, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n-2}, t')$.
2. For each $i \in [n-1]$ check if $\mathsf{verifySigs}(i, (m, t, \sigma_1, \ldots, \sigma_{i-1}), \sigma_i, t)$ is true.
3. If all checks pass, send $(\text{SIGN}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{n-1}))$ to ${\mathcal{F}_{\mathsf{Sig}}}^n$ to obtain $(\text{SIGNATURE}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{n-1}), \sigma_n)$. Output $(\text{SENT}, \mathsf{sid}, m, t, \bar{t} - t, (\sigma_1, \ldots, \sigma_n))$. If a check fails, then output $(\text{NoProof}, \mathsf{sid})$.

Verify: Upon receiving $(\text{VERIFY}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{1o}})$, $\mathcal{V}_i \in \mathcal{V}$ parses $\pi_{\mathtt{1o}} = (\sigma_1, \ldots, \sigma_n)$ and proceeds as follows:

1. Check that $t + \Delta \geq \bar{t}$ and $\mathsf{isP}(t, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,n}, t + \Delta)$ is true.
2. For each $i \in [n]$ check if $\mathsf{verifySigs}(i, (m, t, \sigma_1, \ldots, \sigma_{i-1}), \sigma_i, t)$ is true.
3. If all checks pass set $b = 1$, else $b = 0$. Output $(\text{VERIFIED}, \mathsf{sid}, m, t, \Delta, \pi_{\mathtt{1o}}, b)$.

Tick: Proceed as follows and then send (UPDATE) to $\mathcal{G}_{\mathsf{Clock}}$.

1. Each $\mathcal{P}_i \in \{\mathcal{P}_2, \ldots, \mathcal{P}_{n-1}\}$ sends $(\text{REC}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i-1}$.
2. If $\mathcal{P}_i$ obtains $(\text{REC}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{i-1}), t_{i-1})$ then check if $\mathsf{isP}(t, \mathsf{f}_{\Delta,1}, \ldots, \mathsf{f}_{\Delta,i-2}, t_{i-1})$ is true and if for each $j \in [i-1]$ it holds that $\mathsf{verifySigs}(j, (m, t, \sigma_1, \ldots, \sigma_{j-1}), \sigma_j, t)$ is true.
3. If the checks pass, send $(\text{SIGN}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{i-1}))$ to ${\mathcal{F}_{\mathsf{Sig}}}^i$ to obtain $(\text{SIGNATURE}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_{i-1}), \sigma_i)$ if this is the first message for t.
4. Send $(\text{SEND}, \mathsf{sid}, (m, t, \sigma_1, \ldots, \sigma_i))$ to $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i}$.

Function $\mathsf{verifySigs}(\ell, m, \sigma, t)$**:**

1. Send $(\text{RETRIEVE}, \mathsf{sid}, \mathcal{P}_\ell)$ to $\mathcal{F}_{\mathsf{Reg}}$, receiving $(\text{RETRIEVE}, \mathsf{sid}, \mathcal{P}_\ell, \mathsf{SIG}.vk, t_{Reg})$ as answer. Check that $t_{Reg} \leq t$ and output false if not.
2. Send $(\text{VERIFY}, \mathsf{sid}, m, \sigma, \mathsf{SIG}.vk)$ to ${\mathcal{F}_{\mathsf{Sig}}}^\ell$, receiving $(\text{VERIFIED}, \mathsf{sid}, m, \sigma, f)$ as response. Output true if $f = 1$, otherwise false.

If $\mathcal{P}_1$ is honest then $\mathcal{S}$ already initially obtains m from $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_{\Delta}}$ and honestly generates messages and signatures for $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i}$ where an honest party is a sender. If $\mathcal{P}_1$ is corrupted then wait until the first honest party $\mathcal{P}_i$ obtains the first valid message m, t from $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta},i-1}$. If an honest $\mathcal{P}_i$ would sign and forward the message, then send $(\text{SEND}, \mathsf{sid}, m, t)$ to $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_{\Delta}}$ and continue to simulate the protocol honestly.

Continue simulation for each honest intermediate party until the last honest party $\mathcal{P}_j$. If $\mathcal{P}_j = \mathcal{P}_n$ then $\mathcal{S}$ makes message delivery of $\mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta,n-1}}$ coincide with output delivery in $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ by using **Release message** and chooses the proof string according to all signatures as in the protocol. If some signatures are not valid or delivery appears too late at the simulated $\mathcal{P}_n$ or any honest intermediate receiver then $\mathcal{S}$ makes $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ output (NoProof, sid). Finally, reject all **Verify** queries in case (NoProof, sid) was sent and accept only those for the chosen proof string otherwise. If $\mathcal{P}_j \neq \mathcal{P}_n$, let all honest parties act like in the protocol. For each query of **Verify**, reject if the proof string disagrees with the honestly generated signatures for the specific message and delay. For all signatures of adversarially controlled parties $\mathcal{P}_i$, check with ${\mathcal{F}_{\mathsf{Sig}}}^i$ if they are valid for m, t and only set $\phi = 1$ iff all are valid.

The messages that the adversary obtains in the protocol are perfectly indistinguishable from those in the simulation. Moreover, the output of **Verify** both in the simulation and in the protocol coincides. If the receiver is honest, then delivery of message and output is simultaneous with what happens in the protocol by $\mathcal{S}$ using the **Release message** interface. Moreover the message and its timestamp are consistent with the simulation, and exactly those get delivered to an honest receiver that don't make the protocol abort. If a protocol instead fails, then $\mathcal{S}$ uses (NoProof, sid) to let $\mathcal{F}_{\mathsf{SCD}}^{\mathsf{f}_\Delta}$ abort. All **Verify** responses of $\mathcal{S}$ are consistent with what an honest verifier would output in the protocol. □

C Proof of Theorem 5

Proof. We prove Theorem 5 by constructing a simulator $\mathcal{S}$ (presented in Simulator 19) that executes an internal copy of $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathsf{tlp}}$ in an ideal world execution that is indistinguishable for the environment $\mathcal{Z}$ from the real world execution of $\pi_{\mathsf{TLP-Light}}$ with $\mathcal{A}$. The core tasks of $\mathcal{S}$ are making sure that every puzzle generated by $\mathcal{A}$ in the simulation is created at $\mathcal{F}_{\mathsf{tlp}}$ and that every puzzle that is solved by $\mathcal{F}_{\mathsf{tlp}}$ in the ideal world is simulated towards $\mathcal{A}$. The first task is accomplished by $\mathcal{S}$ by extracting the message m and proof π from every puzzle generated by $\mathcal{A}$ and creating a TLP containing m by contacting $\mathcal{F}_{\mathsf{tlp}}$. The second task is achieved by simulating an execution of $\pi_{\mathsf{TLP-Light}}$ for solving TLPs provided by $\mathcal{F}_{\mathsf{tlp}}$ and later using the leakage of m, π from $\mathcal{F}_{\mathsf{tlp}}$ to program the restricted programmable random oracles such that the output of the protocol matches m, π. Both simulation strategies are clearly possible and indistinguishable from a real execution since $\mathcal{S}$ has the shared secret key sk provided by $\mathcal{F}_{\mathsf{DKG}}$ (which is simulated) and since it can rely on the properties of the IND-CCA secure (and thus UC-secure) encryption scheme in Definition 5, which is used to generate ciphertexts containing TLP messages in $\pi_{\mathsf{TLP-Light}}$. □

D UC Treatment of Delay Encryption

The notion of Delay Encryption (DE) was introduced in [15], where a game based security definition is presented. In order to use our proof of sequential

Simulator 19: $\mathcal{S}$ for $\pi_{\mathsf{TLP-Light}}$

$\mathcal{S}$ interacts with an internal copy of $\mathcal{A}$, towards which it simulates the honest parties in $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and functionalities $\mathcal{G}_{\mathsf{ticker}}, \mathcal{G}_{\mathsf{Clock}}, {\mathcal{G}_{\mathsf{rpoRO}}}^1, {\mathcal{G}_{\mathsf{rpoRO}}}^2, \mathcal{F}_{\mathsf{DKG}}, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta,1}}, \ldots, \mathcal{F}_{\mathsf{mdmt}}^{\mathsf{f}_{\Delta,|\mathcal{W}|-1}}$. Unless explicitly stated, $\mathcal{S}$ simulates all functionalities exactly as they are described.

Setup: $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{DKG}}$ towards $\mathcal{A}$ and honest parties in $\mathcal{P}$ interacting with $\mathcal{F}_{\mathsf{DKG}}$, learning all sk_j and $\mathsf{sk} = \sum_{\mathcal{P}_j \in \mathcal{W}} \mathsf{sk}_j$.

Create puzzle: When $\mathcal{A}$ outputs $\mathsf{puz} = (c_1, c_2, c_3)$, $\mathcal{S}$ proceeds as follows:

1. Extract the message m and proof $\pi = (\mathsf{pk}, r, s)$: (a) Extract message m by computing $r = \tilde{c}_2 = c_2 \cdot c_1^{\mathsf{sk}}$, sending (Hash-Query, r) to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$, receiving (Hash-Confirm, pad) and computing $m|s = c_3 \oplus \mathsf{pad}$. (b) Check that the puzzle is valid by sending (Hash-Query, $m|s$) to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$, receiving (Hash-Confirm, ρ) and checking that $\mathsf{puz} = (c_1 = g^\rho, c_2 = r \cdot \mathsf{pk}^\rho, c_3 = (m|s) \oplus \mathsf{pad})$. (c) Send (IsProgrammed, $m|s$) (resp. (IsProgrammed, r)) to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$ (resp. ${\mathcal{G}_{\mathsf{rpoRO}}}^2$) and abort if either of the responses is (IsProgrammed, 1).
2. If all checks on m, π passed, send $(\mathsf{CreatePuzzle}, \mathsf{sid}, m)$ to $\mathcal{F}_{\mathsf{tlp}}$ and provide puz, π when requested.

Solve: Simulate honest parties in $\mathcal{P}$ executing as in $\pi_{\mathsf{TLP-Light}}$. Upon receiving $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ from $\mathcal{F}_{\mathsf{tlp}}$, $\mathcal{S}$ forwards $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ to the first $\mathcal{P}_i \in \mathcal{W}$.

Public Verification: Simulate honest parties in $\mathcal{P}$ executing as in $\pi_{\mathsf{TLP-Light}}$.

Tick: $\mathcal{S}$ simulates honest parties in $\mathcal{W}$ executing as in $\pi_{\mathsf{TLP-Light}}$, additionally performing the following steps:

Starting Solution: When a corrupted party in $\mathcal{P}$ sends $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ to the first $\mathcal{P}_i \in \mathcal{W}$, $\mathcal{S}$ forwards $(\mathsf{Solve}, \mathsf{sid}, \mathsf{puz})$ to $\mathcal{F}_{\mathsf{tlp}}$.

Ongoing Solution: $\mathcal{S}$ answers requests from $\mathcal{F}_{\mathsf{tlp}}$ as follows:

- Upon receiving $(\mathsf{Solved}, \mathsf{sid}, \mathsf{puz}, m, \pi)$ from $\mathcal{F}_{\mathsf{tlp}}$, $\mathcal{S}$ programs ${\mathcal{G}_{\mathsf{rpoRO}}}^1$ and ${\mathcal{G}_{\mathsf{rpoRO}}}^2$ such that solving puz via the steps of $\pi_{\mathsf{TLP-Light}}$ yields message m with proof π.
- Upon receiving a request from $\mathcal{F}_{\mathsf{tlp}}$ for π for a $\mathsf{puz} = (c_1, c_2, c_3)$, $\mathcal{S}$ answers with $\pi = (\mathsf{pk}, r, s)$ obtained by computing $r = \tilde{c}_2 = c_2 \cdot c_1^{\mathsf{sk}}$, sending (Hash-Query, r) to ${\mathcal{G}_{\mathsf{rpoRO}}}^1$, receiving (Hash-Confirm, pad) and computing $m|s = c_3 \oplus \mathsf{pad}$.

communication delay machinery, we first introduce a treatment of DE in the UC framework, upon which we have defined and constructed our results. In Functionality 20, We provide an ideal functionality $\mathcal{F}_{\mathsf{DE}}$ for DE that captures this notion.

Similarly to other timed functionalities in our work, this functionality is defined in the abstract composable time model of TARDIS [8], previously discussed in Section 2 and Appendix A.2. We essentially adapt our PV-TLP functionality $\mathcal{F}_{\mathsf{tlp}}$ to generate a DE ciphertext as if it was a time-lock puzzle connected to a certain ID represented by a sub-session ID ssid. Analogously, we modify the puzzle solving interface to instead implicitly extract the secret key corresponding to a ssid, which in the functionality is reflected by allowing honest parties to obtain

the messages in ciphertexts corresponding to that ssid. As is the case in $\mathcal{F}_{\mathsf{tlp}}$ and $\mathcal{F}_{\mathsf{VDF}}$, we allow the adversary to decrypt ciphertexts connected to a given ssid slightly before the same access is given to honest parties (*i.e.* at time $\epsilon\Gamma < \Gamma$).

Functionality 20: $\mathcal{F}_{\mathsf{DE}}$

$\mathcal{F}_{\mathsf{DE}}$ is parameterized by a computational security parameter τ, a message space $\mathcal{MSG}$, a tag space $\mathcal{TAG}$, a slack parameter $0 < \epsilon \leq 1$ and a delay parameter Γ. $\mathcal{F}_{\mathsf{DE}}$ interacts with a set of parties $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{DE}}$ maintains initially empty lists omsg (encrypted messages), L (keys being extracted), EXT (extracted keys).

Encrypt Message: Upon receiving a message $(\mathsf{CreatePuzzle}, \mathsf{sid}, \mathsf{ssid}, m)$ from $\mathcal{P}_i$ where $m \in \mathcal{MSG}$, send $(\mathsf{CreatePuzzle}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$ and let $\mathcal{S}$ provide puz. If $\mathsf{puz} \notin \mathcal{TAG}$ or there exists $(\mathsf{ssid}, \mathsf{puz}, m') \in \mathsf{omsg}$, then $\mathcal{F}_{\mathsf{DE}}$ halts. Append $(\mathsf{ssid}, \mathsf{puz}, m)$ to omsg, set and output $(\mathsf{Encrypt}, \mathsf{sid}, \mathsf{ssid}, \mathsf{puz})$ to $\mathcal{P}_i$ and to $\mathcal{S}$.

Extract Key: Upon receiving $(\mathsf{Extract}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{P}_i \in \mathcal{P}$, add $(\mathsf{ssid}, 0)$ to L and send $(\mathsf{Extract}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$.

Decrypt Ciphertext: Upon receiving $(\mathsf{Decrypt}, \mathsf{sid}, \mathsf{ssid}, \mathsf{puz})$ from a party $\mathcal{P}_i \in \mathcal{P}$, ignore the message if $\mathcal{P}_i$ is honest and there does not exist a record $\mathsf{ssid} \in \mathsf{EXT}$ or if $\mathcal{P}_i$ is corrupted and there does not exist a record $(\mathsf{ssid}, c) \in L$ for $c \geq \epsilon\Gamma$. Otherwise, proceed as follows:

- If $(\mathsf{ssid}, \mathsf{puz}, m) \in \mathsf{omsg}$, output $(\mathsf{Decrypt}, \mathsf{sid}, \mathsf{ssid}, \mathsf{puz}, m)$ to $\mathcal{P}_i$.
- If there does not exist $(\mathsf{ssid}, \mathsf{puz}, m) \in \mathsf{omsg}$, let $\mathcal{S}$ provide $m \in \mathcal{MSG}$, add $(\mathsf{ssid}, \mathsf{puz}, m)$ to omsg and output $(\mathsf{Decrypt}, \mathsf{sid}, \mathsf{ssid}, \mathsf{puz}, m)$ to $\mathcal{P}_i$.

Tick: For all $(\mathsf{ssid}, c) \in L$, update $(\mathsf{ssid}, c) \in L$ by setting $c = c + 1$ and:

- If $c \geq \epsilon\Gamma$, send $(\mathsf{Extracted}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$.
- If $c = \Gamma$, remove $(\mathsf{ssid}, c) \in L$, send $(\mathsf{Proceed?}, \mathsf{sid}, \mathsf{ssid})$ to $\mathcal{S}$ and proceed as follows:
  - If $\mathcal{S}$ sends $(\textsc{Abort}, \mathsf{sid}, \mathsf{ssid})$, output $(\mathsf{Abort}, \mathsf{sid}, \mathsf{ssid})$ to all $\mathcal{P}_i$.
  - If $\mathcal{S}$ sends $(\textsc{Proceed}, \mathsf{sid}, \mathsf{ssid})$, add ssid to EXT and output $(\mathsf{Extracted}, \mathsf{sid}, \mathsf{ssid})$ to all $\mathcal{P}_i$.

E Practical Considerations

In this appendix, we elaborate on our model choices and how realistic our constructions are in generic terms. Unfortunately we cannot back our estimates with concrete results as we could not buy a few satellites, ship them to space and test our protocol in its realistic setting. We leave this as interesting future work.

How to compute communication delay lower bounds. In Physics c denotes the speed of light (measured to $c = 299.792.458$ meters/second in the space). Einstein's Special Relativity sets c as the natural upper bound on communication speed since matter, energy or signals that may carry information

can travel at most as fast as the speed of light. With this in mind it is straightforward to determine the exact lower bound for communication delay between two satellites. Let d denote the distance in meters between two satellites, then the minimal possible time-delay in their communication is $\Delta = d/c$. The distance d can be computed by first determining each satellite's position and then computing the Euclidean distance between such positions. Determining a satellite's position at an instant in time is done via classical mechanics, see [5, Chapter 4 & 5] or [44, Chapter 10 & 11] for standard references. Even spy satellites can be tracked by amateur enthusiasts, e.g. `https://gizmodo.com/how-you-can-track-every-spy-satellite-in-orbit-1685316357`.

Efficiency of TLP & IBE constructions. When computing the Time-Lock Puzzle (TLP) based on threshold encryption, each satellite performs one extra scalar multiplication, adding 0.066ms for the Cortex-A15 processor and 2.28ms for the A9 processor mentioned above. When executing our VDF/TLP constructions based on IBE, each satellite only needs to compute one extra scalar multiplication on the elliptic curve as in the TLP based on threshold encryption. Expensive operations (e.g. re-encryption and bilinear pairings) are only done on non-constrained devices verifying the result of VDF/TLP evaluations.

On a trust assumption. Previous results on TLPs/VDFs consider that the evaluation of TLPs/VDFs is done locally by each party, thus requiring security even when this single evaluator is dishonest. In our setting, we outsource this evaluation to a group of parties and guarantee security if at least 1 of them is honest. In our concrete instantiation, we require at least one of the parties signing the message be honest, when the message travels through the round-robin network of parties when being signed in order by each party. While this is indeed an extra trust assumption, it allows us to provide precise and absolute delay lower bounds. This is not unprecedented in the time-based cryptography literature, as the same assumption of at least 1 out of n parties being honest is also made in the context of distance bounding protocols. Moreover, since satellites are in orbit, it is infeasible to corrupt their hardware and software (provided it is not updatable) after the launch.

Liveness of Optimistic Protocols. We take an optimistic approach of designing highly efficient protocols that might abort in case of misbehavior by one of the parties, in which case we resort to more expensive protocols that identify and eliminate the cheating party. This applies to our constructions of VDFs in Section 5,TLPs in Section 7 and Delay Encryption in Section 6. All of the constructions rely on our proof of communication delay, so they will abort if a satellite in the pre-established signing path fails to provide a valid signature. Moreover, in the TLP (resp. Delay Encryption) constructions, a satellite who misbehaves in the threshold encryption (resp. threshold identity key generation) will also cause an abort. Both abort cases can be handled by requiring the satellites to repeat the protocol while providing non-interactive zero knowledge proofs

(NIZK) of correct execution. In this augmented protocol, we can easily identify a cheater by checking the NIZKs (*i.e.* misbehavior will result in an invalid NIZK), subsequently eliminating this cheater e re-executing the protocol once more. Naturally, eliminating a cheating satellite will also require re-executing the sequential signing protocol, which might be costly. However, notice that once a cheater is eliminated, it no longer participates in future executions of the protocol. Hence, these re-executions will happen at most t times, where t is the number of corrupted satellites. After all cheaters are eliminated, all executions will only require the highly efficient optimistic protocol.