ChronoCloak: An Integrated Solution for Mitigating Premature Disclosure in Oblivious Digital Dissemination

Ahmed Zawia(✉) and M. Anwar Hasan

University of Waterloo, Waterloo, ON, Canada
{azawia,ahasan}@uwaterloo.ca

Abstract. With the increasing use of online digital content delivery, such as games, videos, and magazine articles, there are scenarios where it is crucial to prevent premature revelation of the distributed content and, at the same time, allow a recipient to choose any specific item of the content bundle in a privacy-preserving manner. In this work, we propose an integrated solution, namely ChronoCloak, which aims to address both the premature exposure of the sender data and the privacy preservation of the receiver interaction. ChronoCloak allows a sender to transmit a set of secrets through a puzzle, which can be solved via a lengthy computation. Upon solving the puzzle, the receiver recovers only a subset of the secrets that is oblivious to the sender. It also allows the receiver to securely outsource the computation (with public verifiability), yet only the intended receiver can retrieve a subset of secrets using the puzzle solution. We also propose an ideal functionality for ChronoCloak and provide a generic construction implementing this functionality in the random oracle model, using an ideal oblivious transfer functionality and a time lock-like function.

Keywords: Delay-based cryptography · Oblivious transfer · Trapdoor verifiable delay functions · Time-lock puzzles

1 Introduction

In certain cases of digital content distribution, it is crucial for the distributor to be able to send a selection of time-locked items (e.g., games, videos, or magazine articles) to a recipient who can then choose and unlock one of those items after a predetermined time in a privacy preserving manner so that the distributor is unaware of the recipient's choice. For example, the publisher of a weekly magazine may send digital copies of encrypted articles of an upcoming issue to its reader base in advance. Then, a *pay-per-article* subscriber, on or after the publication date, can decrypt one of the articles of their choice that is oblivious to the publisher. The core issues, namely time-locking of contents and their oblivious transfer, in the aforementioned applications can potentially be addressed by carefully deploying relevant cryptographic primitives and other necessary

N. Mouha and N. Nikiforakis (Eds.): ISC 2024, LNCS 15257, pp. 232–251, 2025.
https://doi.org/10.1007/978-3-031-75757-0_12

functional blocks. For example, one existing solution is a time-release oblivious transfer employing a verifiable ID-based encrypted blind signature [45], where a trusted third party (time server) is used to ensure the release of the message at a predetermined time in the future. Time tokens can be broadcast periodically only if the server is online. In the work of [36], privacy is guaranteed only for a limited period of time. As in [36,45] uses a time server to release a private key that enables the sender to learn the receiver's choice after a specified amount of time has passed. A second solution can be constructed using [8]'s framework, which is an inefficient method of constructing similar functionality using multiple primitives. However, no concrete construction has been proposed. As trusted setup can be difficult to perform securely, using multiple primitives as a solution may require multiple calls to a trusted setup. This significantly complicates the system setup process thereby increasing potential security risks. We then ask: if both issues occur simultaneously in a system, how can we address them in a single solution?

Contributions. In this work, we consider an integrated approach to address time-delayed decryption and oblivious transfer and propose a solution called *ChronoCloak*. Through ChronoCloak, the sender and receiver interact via secret trapdoors and construct a joint puzzle. After solving the puzzle via a lengthy computation (i.e., a sequential computation of length $T \in \mathbb{N}$), a subset of the sender's secrets can be obliviously retrieved by the receiver. More importantly, our ChronoCloak allows the recipient to outsource the lengthy computation to an external solver without worrying that the latter can decrypt the sender's secret. This is because the puzzle is jointly generated by both the sender's and the intended recipient's secret trapdoors and, therefore, recovering the transmitted secret requires more than just the solution to the puzzle - it also requires the recipient's secret trapdoor. In this article, we introduce an ideal functionality for ChronoCloak and present a generic construction that implements this functionality in the random oracle model, integrating an ideal oblivious transfer functionality along with a time lock-like function.

Related Work. As discussed previously, in the literature there are proposals for constructing Oblivious Transfer (OT) protocols with time-release delays, among which [21,36,45] are the most pertinent. The problem of sending messages into the future with conditional oblivious transfer was considered by Crescenzo et al. [21]. In the case of delayed release, the authors of [45] incorporate a trusted third party as a time server to ensure the scheduled release of the message. In the work of [36], however, the receiver's choice privacy is maintained within a constrained time frame. Similar to [45], the work in [36] uses a time server to release a private key that allows the sender to learn the receiver's choice after a specified time. As the time server generally requires output some secrets at every time interval, the reliability and security requirement for the protocol's trusted-party extends beyond the one-time setup of initializing the protocol parameters.

Delay-Based Primitives. Aside from time servers, several cryptographic primitives have been proposed to guarantee time delay, such as time-lock puzzles (TLP) [41], proof of sequential work [1,20,33,37], and verifiable delay functions (VDF) [17,25,35,39,44] (and its relative trapdoor VDF [44,46]). Furthermore,

the security treatment of these primitives lacks composability guarantees, and thus, integrating them securely into more complex protocols is not intuitive. This shortfall can be addressed by modeling them in the universal composability framework [12]. There are also several studies that discuss maintaining composability in delay-based primitives, including [3,8,9,24,26,30], some with and some without public verifiability.

Oblivious Transfer. The concept of oblivious transfer was initially introduced by [40]. Since then, it has been a crucial building block in many secure multi-party computations [18,29]. Over time, different constructions of OT protocols have been developed based on different assumptions. One notable example is the Chou-Orlandi OT scheme [19], which is a three-round OT based on Diffie-Hellman (DH) key exchange. Although the Chou-Orlandi scheme did not achieve UC security [7,27,34], their core idea inspired the design of secure and more efficient OT protocols such as [4,14,31,42]. Also, there are a number of isogeny-based oblivious transfer constructions, including [2,5,6,32,42,43]. In [2], the authors propose a framework for developing cryptographic primitives which is amenable to group-actions such as CSIDH [16] and CSI-FiSh [10]. Using their framework, the authors describe the construction of a variety of cryptographic primitives, including an (inefficient) statistically sender-private OT protocol. A notable work by Lai et al. [31] introduced efficient OT construction based on the *reciprocal* CSIDH assumption, using a *quadratic twist map* to reduce a 3-round Chou-Orlandi OT scheme to 2 rounds. They incorporated a "proof of decryption" (PoD) mechanism, achieving UC security. Their PoD assumption, however, had a security flaw, which was rectified in the revised version [32], resulting in a four-round OT scheme. An optimized OT extension introduced in [5] reduces isogeny computations, resulting in a weaker notion of OT security, i.e., OT with selective failure attack. Additionally, they proposed two optimal OTs with 4 and 2 rounds for different models. On the other hand, we note that some of the isogeny-based schemes such as [6,42,43] have been affected by recent attack [15] on SIDH, but there is no known way to extend such attacks to the general isogeny problem. There are several other assumptions that can be used to construct oblivious transfer such as lattice-based [38], and code-based [11,22,23].

2 Background

Notations. We use $A \| B$ to denote the concatenation of a string representation of A and B. We use the calligraphic font to denote a finite set (e.g., $\mathcal{B}$). The size of a set $\mathcal{B}$ is represented by $|\mathcal{B}|$. We denote the process of uniformly sampling a random element e from $\mathcal{B}$ by $e \leftarrow_{\$} \mathcal{B}$, whereas the deterministic choice of an element e from $\mathcal{B}$ is referred to by $e \leftarrow \mathcal{B}$. The process of executing an algorithm Alg on a uniformly random distribution is denoted by $a \leftarrow_{\$} \mathsf{Alg}$, where a is the output. On the other hand, the deterministic process of executing an algorithm Alg is denoted by $a \leftarrow \mathsf{Alg}$. A function's composition is denoted by $\circ$ such that $\mathsf{f}_1 \circ \mathsf{f}_2(x) = \mathsf{f}_1(\mathsf{f}_2(x))$ for a valid input x. The set $\{1, \ldots, n\}$ is denoted by $[n]$ for a positive integer n. The process of sampling the vector $(z_i)_{i \in [n]}$ of size n

for a uniform distribution on $\mathcal{B}^n$ is denoted by $\boldsymbol{z} \leftarrow_{\$} \mathcal{B}^n$ such that $z_i \leftarrow_{\$} \mathcal{B}$ for all $i \in [n]$. $\mathsf{Pr}[\mathsf{Ev} : \mathsf{Ev}_1, \mathsf{Ev}_2, \ldots, \mathsf{Ev}_n]$ refers to the probability of the event Ev arising after orderly events $\mathsf{Ev}_1, \mathsf{Ev}_2, \ldots, \mathsf{Ev}_n$. Finally, a scheme's security level is represented by $\lambda \in \mathbb{N}$.

2.1 Overview of the Security Model

The following is a brief overview of the security model used in our protocol, based on universally composable (UC) security [12]. We then present related definitions, including those for oblivious transfer and the random oracle.

Functionality. In the UC framework, the ideal functionality F acts as a trusted third party, e.g., two party computations involve F that maps inputs to outputs, is defined as $\mathsf{F} = (\mathsf{F}_1, \mathsf{F}_2) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$. The first party, whose input is x_1, wants the output $\mathsf{F}_1(x_1, x_2)$, and the second party, whose input is x_2, wants the output $\mathsf{F}_2(x_1, x_2)$. The security of the model relies on the fact that each party can only compute the output for their own input, without knowing the other party's input. **Static corruption:** In static corruption, corrupted parties remain corrupted during protocol execution, while honest parties remain honest. **Distinguisher** Suppose Env is the environment entity that decides parties' inputs (including the adversary's auxiliary input) and receives all parties' outputs, Env serves as an interactive distinguisher between two distributions.

Malicious Adversarial Model. A malicious adversary A has no restrictions on their actions-it can deviate from the construction specification, changes its input, and alters its strategy. **Real Execution** Parties interact directly to carry out construction execution. **Ideal execution** Parties interact through the construction's intended ideal functionality, F. **Hybrid Execution** It involves parties interacting not only with each other, as in real execution, but also with a trusted entity computing functionalities (e.g., F_{RO}) over which a simulator ($\tilde{\mathcal{A}}$) has some control. Briefly, in the UC framework, after parties have obtained their inputs, the honest party sends its private input to F, and a simulation sends the adversary's input to F. The simulation interacts with the adversary (corrupting a party P_i) as a blackbox; hence, it needs to extract the input from the adversary through normal construction's rounds of interaction, and passes it to F. In the following steps, F provides the adversary's output to the simulator, which in turn passes it on to the adversary to complete the construction rounds (for more information, see [28, Section 2.3.1]). Upon notification by the simulator, it is then F's responsibility to release the honest party's output. In this work, we assume that the simulator $\tilde{\mathcal{A}}$ is probabilistic polynomial-time. Generally speaking, as an interactive distinguisher, Env interacts with the adversary during the execution to ultimately distinguish between the real execution and the ideal execution. Furthermore, the constructions in this article will be presented in a hybrid model due to the limitations of the UC framework outlined by Canetti et al. [13].

Definition 1. *Against malicious adversaries, a construction Γ securely implements a deterministic functionality F if*

$$\exists \text{ a simulator } \tilde{\mathcal{A}} \text{ s.t. } \mathit{Hybrid}^{\hat{\mathsf{F}}}_{\Gamma,\mathcal{A},\mathit{Env}} \stackrel{c}{=} \mathit{Ideal}_{\mathsf{F},\tilde{\mathcal{A}},\mathit{Env}},$$

for any probabilistic polynomial-time adversary A *and distinguisher* Env*, where* $\mathsf{Ideal}_{F,\tilde{\mathcal{A}},Env}$ *represents all parties' output distribution ensemble from the ideal execution of* F *(in which one of the parties is corrupted by* $\tilde{\mathcal{A}}$*), and* $\mathsf{Hybrid}^{\hat{F}}_{\Gamma,A,Env}$ *represents all parties' output ensemble from the real execution in which parties (one of whom is corrupted by* A*) have access to a functionality* $\hat{\mathsf{F}}$ *(e.g.,* F_{RO}*).*

Related Definitions

Oblivious Transfer Protocol, F_{OT}. A 1-out-of-2 oblivious transfer protocol allows for the transmission of one of two messages obliviously. This protocol involves a trusted setup party, a sender (P_S), and a receiver (P_R), typically requiring n rounds of interaction. The sender holds two secret messages, m_0 and m_1, while the receiver has a choice bit i. Through n rounds of interaction, the sender and receiver transfer a secret message, allowing the receiver to reconstruct the selected message, m_i. In Fig. 1, we present the ideal functionality of 1-out-of-2 oblivious transfer protocol, namely F_{OT}.

Commit: Upon receiving (Commit, *sid*, P_S, $i \in \{0,1\}$) from P_R, check if a (P_R, P_S, *sid*, $i \in \{0,1\}$) was previously stored. If yes, ignore; otherwise, store (P_R, P_S, *sid*, $i \in \{0,1\}$) and output (Commit, *sid*, P_R) to P_S.
Challenge: Upon receiving (Challenge, *sid*, P_R, m_0, m_1) from P_S, check if a (P_R, P_S, *sid*, i) was previously stored. If not, ignore; otherwise, send (Challenge, *sid*, P_S, m_i) to P_R and (Challenge, *sid*, P_R) to P_S.

Fig. 1. Oblivious transfer ideal functionality (F_{OT}).

Random Oracle Functionality, F_{RO}. Below we present F_{RO}, which initiates with an empty record.

Initiate: Upon receiving (Initiate, *sid*) from $\tilde{\mathcal{A}}$, setup an empty record $\mathbf{H}$, representing the state of $\mathsf{F}_{\mathsf{RO}} : \{0,1\}^* \to \mathcal{H}$; and no further messages of this type is accepted.
Query: Upon receiving (Query, *sid*, inString $\in \{0,1\}^*$) from participant P, return (Query, *sid*, $str \in \mathcal{H}$) as follows: (i) If (-, inString) $\notin \mathbf{H}$ (i.e., previously, there is no record containing an entry inString in $\mathbf{H}$), sample $str \leftarrow_{\$} \mathcal{H}$ and then add the tuple (str, inString) to $\mathbf{H}$. (ii) If there is a record containing inString in $\mathbf{H}$, retrieve str from $\mathbf{H}$ (i.e., retrieve (str, inString) $\leftarrow \mathbf{H}$).

Fig. 2. The random oracle ideal functionality (F_{RO}).

3 Underlined UC Framework and Operation Unit

In this section, we present an overview of the underlined UC framework. We first present an informal definition related to a generic sequential function. Following this, we highlight the concept of capturing the abstract progression of computations through a series of ticks from a global ticker $\mathsf{F}_{\mathsf{clk}}$. To capture computation costs (expressed in ticks from $\mathsf{F}_{\mathsf{clk}}$), we then introduce an *operation unit* (OU) and its ideal functionality (F_{OU}), which each party accesses to execute a protocol's algorithms.

Long Sequential Functions. We use T to characterize a function's difficulty, representing the amount of sequential work required to compute its output for any random input with a polynomially large number of parallel computations. A short function has a small T, while a function with a large T is considered a long function. Throughout this work, we refer to the long sequential function as $\mathsf{SeqEval}$. $\mathsf{SeqEval}$ is defined with parameters for its domain and range. A shorter version of $\mathsf{SeqEval}$, referred to as a shortcut or trapdoor, is denoted by tr.

UC Framework and a Global Ticker. As depicted in Fig. 3, our framework relies on the ideal functionality of a global ticker ($\mathsf{F}_{\mathsf{clk}}$) from [9], providing "ticks" (representing a unit of time) on a one-at-a-time basis. $\mathsf{F}_{\mathsf{clk}}$ is designed to capture the units of passing time and starts with an empty record of enrolled functionalities and parties. Our work implicitly assumes that all relevant functionalities and parties of a protocol are enrolled with $\mathsf{F}_{\mathsf{clk}}$.

Enrollment: Upon receiving Enroll from a party P (resp. from functionality F), add P to $\mathbf{P}$ (resp. add F to $\mathbf{F}$); then send back Enrolled.
Ticked request: Upon receiving IsTicked from $\mathsf{F} \in \mathbf{F}$ and $\mathsf{F} \notin L_{\mathsf{F}}$, add F to L_{F}, and send ticked to F; otherwise, send NotTicked to F.
Ready: Upon receiving Ready from $\mathsf{P} \in \mathbf{P}$, add P to L_P, and send Success to P.
Tick: Upon receiving tick from Env, if $\mathbf{P} = L_p$, set both $L_p = \varnothing$ and $L_{\mathsf{F}} = \varnothing$, and send ticked to the adversary $\tilde{\mathcal{A}}$. Otherwise, send NotTicked to Env.

Fig. 3. The functionality of $\mathsf{F}_{\mathsf{clk}}$-global ticker from [9].

$\mathsf{F}_{\mathsf{clk}}$'s ticks are initiated by the environment Env's query Tick, but $\mathsf{F}_{\mathsf{clk}}$ will only tick if all parties have sent (in an arbitrary order) a Ready query. Moreover, to emulate a delay based on computation, parties do not (directly) receive notification of ticks (excluding simulator $\tilde{\mathcal{A}}$). They rather discretely query the relevant functionalities, which in turn respond appropriately to each query based on its internal state. Upon query such as Elicitate, functionalities first check (via IsTicked) with $\mathsf{F}_{\mathsf{clk}}$ to determine if a tick has occurred. In such a case, the functionality Tick interface will be triggered internally. Then the functionality will return an appropriate message (on request) to the relevant party(ies). As soon as

all relevant functionalities have responded, the party will submit a Ready query to $\mathsf{F}_{\mathsf{clk}}$, requesting an advance to the next tick. For clarity, we assume that the query messages to/from $\mathsf{F}_{\mathsf{clk}}$ are handled implicitly in our protocol due to their repetitiveness.

Generally, the environment is capable of conducting computations instantaneously relative to other parties, e.g., it can perform computation before activating honest parties. To effectively simulate time delay and to align with a more realistic model, we propose a global operation unit OU, mandating that all parties utilize this OU for conducting any form of arbitrary computation. This OU operates under tick-based functionality, governed by $\mathsf{F}_{\mathsf{clk}}$, ensuring no party, including the environment, can advance computationally ahead of others; thereby constraining the environment's advantage.

Modeling Operation Unit. Here we introduce an operation unit OU with a finite instruction set (INST). OU is formally defined as follows: $\mathsf{OU} : \mathsf{INST} \times \mathsf{STR} \rightarrow \mathsf{STR}$, where STR is the input/output space. Each party P has *private access* to OU, through which P executes algorithm steps. To execute an algorithm, P translates the steps into a sequence of instructions (i.e., a sequence of instructions executing a step S is referred to as $\mathbf{Inst}_{\mathrm{S}} = \{\mathsf{Inst}_j\}_{j\in[k]}$ s.t. $k \in \mathbb{N}$ and all $\mathsf{Inst} \in \mathsf{INST}$) and parses the input in STR's format. P has the flexibility to construct $\mathbf{Inst}_{\mathrm{S}}$ arbitrarily (of polynomial size), reflecting the strategy used to achieve the desired output.

OU's execution model: Let τ_F be a *global* function that takes $\mathbf{Inst}_{\mathrm{S}}$ as input and outputs an integer $\tau \in \mathbb{N}$, representing the number of ticks required for $\mathbf{Inst}_{\mathrm{S}}$'s execution. To emulate computation costs (in ticks), we introduce the ideal functionality F_{OU}, which captures the process of executing OU described earlier. To mimic OU's execution cost, F_{OU} computes the instruction sequence $\mathbf{Inst}_{\mathrm{S}}$ instantly on a given input in but withholds the output until $\tau = \tau_F(\mathbf{Inst}_{\mathrm{S}}, in)$ ticks have elapsed. As depicted in Fig. 4, to initiate a session with F_{OU}, P sends (Initiate, sid) to F_{OU}, which creates two empty lists, namely $\mathbf{L}_{in}$ and $\mathbf{L}_{out}$. To perform an execution, P submits (Evaluate, sid, $\mathbf{Inst}_S$, in) to F_{OU}. Promptly, F_{OU} evaluates the instruction sequence, storing the query and the execution output in $\mathbf{L}_{in}$, then it informs P with τ. At every tick, for every query in $\mathbf{L}_{in}$, if $\tau = 0$, move the query from $\mathbf{L}_{in}$ to $\mathbf{L}_{out}$, otherwise it decreases τ by one. After τ ticks have passed, P sends Elicitate query to F_{OU}, which returns all queries of P, containing the execution outputs. To simplify the notations, for evaluating Solve.SeqEval, P simply sends (Evaluate, sid, SeqEval, $\hat{in} := \mathbf{c} \parallel \mathrm{RandSeed}$) to F_{OU}, instead of submitting a set of explicit instructions as (Evaluate, sid, $\mathbf{Inst}_{\mathsf{SeqEval}} \leftarrow \mathsf{P}(\mathsf{SeqEval})$, $in \leftarrow \mathsf{P}(\hat{in})$) to F_{OU}. In light of the above discussion, we can define T to be $\tau_F(\mathsf{SeqEval}, \mathbf{c})$.

Initiate: Upon receiving (Initiate, *sid*) from P_j, store *sid* and initiate two empty lists, namely $\mathbf{L}^j_{in}$ and $\mathbf{L}^j_{out}$. Output (Success, *sid*) to P_j.
Evaluate: Upon receiving (Evaluate, *sid*, $\mathbf{Inst}_S$, $in \in \mathsf{STR}$) from P_j, compute $out \leftarrow \mathsf{OU}(\mathbf{Inst}_S, in)$ and $\tau = \tau_F(\mathbf{Inst}_S, in)$. Add ($\mathsf{P}_j$, $\mathbf{Inst}_S$, *sid*, τ, *out*) to $\mathbf{L}^j_{in}$, and Send (Success, *sid*, $\mathbf{Inst}_S$, τ) to P_j.
Tick: In every tick, and for every query in $\mathbf{L}^j_{in}$, if $\tau > 0$, then decrease τ by one and update the query. If $\tau = 0$, then move it from $\mathbf{L}^j_{in}$ and add it to $\mathbf{L}^j_{out}$.
Elicitate: Upon receiving (Elicitate, *sid*) from P_j, pop all entries in $\mathbf{L}^j_{out}$ with (P_j, $\cdots$) and send (Elicitate, *sid*, **resp**) to P_j, where **resp** denotes the popped entries.

Fig. 4. The ideal functionality of the operation unit, F_{OU}.

4 ChronoCloak

Here, we present a formal definition of ChronoCloak and its desired properties.

Definition 2 (ChronoCloak). *ChronoCloak is a tuple of algorithms (Setup, Commit, Challenge, Solve, Verify, Open) defined as follows, where $\mathcal{M}$, $\mathcal{S}$, $\mathcal{C}$, $\mathcal{Y}$, and Π refer to the message, secret, challenge, answer, and proof spaces, respectively.*

- *Setup: a randomized algorithm that takes a security parameter λ and a difficulty T. It runs in time Poly(λ) and outputs a public parameter pp.*
- *Commit: a randomized algorithm that takes pp and $i \in \{0,1\}$; it selects a random secret $sk_r \leftarrow_{\$} \mathcal{S}$, and generates and outputs, in time Poly(λ), a commitment $\boldsymbol{c}'$ of i, and sk_r.*
- *Challenge: a randomized algorithm with input pp, $(m_0, m_1) \in \mathcal{M}^2$, and $\boldsymbol{c}' \in \mathcal{C}$. It selects $sk_c \leftarrow_{\$} \mathcal{S}$, and computes and returns, in time Poly(λ), a challenge $\boldsymbol{c} \in \mathcal{C}$ of $\boldsymbol{c}'$.*
- *Solve: an algorithm that runs in time T with Poly(λ) parallel processors. It takes as an input pp and a challenge $\boldsymbol{c} \in \mathcal{C}$; it returns an answer $\boldsymbol{a} \in \mathcal{Y}$ and a proof $\pi \in \Pi$.*
- *Verify: a deterministic algorithm that receives pp and $\boldsymbol{c} \in \mathcal{C}$ with its proposed answer $\boldsymbol{a} \in \mathcal{Y}$ and proof $\pi \in \Pi$. In time polynomial in $\log T$ and λ, it returns* ACCEPT *if $\boldsymbol{a}$ is the correct unique answer to $\boldsymbol{c}$ with respect to π, otherwise it returns* REJECT.
- *Open: a deterministic algorithm that runs in time Poly($\log T, \lambda$). It takes as an input pp, sk_r, and a challenge $\boldsymbol{c} \in \mathcal{C}$ with its answer $\boldsymbol{a} \in \mathcal{Y}$ and proof $\pi \in \Pi$; with the knowledge of $sk_r \in \mathcal{S}$, it returns $m_i \in \{m_0, m_1\}$ if $\boldsymbol{a}$ is the unique answer to $\boldsymbol{c}$ regarding π, otherwise it returns $\perp$.*

ChronoCloak Properties. Let $\mathcal{A}$ be adversary represented by a set of polynomially bounded algorithms (or/and strategies, which are both referred to by A), e.g., $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2)$. Let A_1 be an offline algorithm, running on time $Poly(T, \lambda)$ that

outputs pc a pre-computation of pp. Furthermore, it is the internal coin tosses of ChronoCloak's algorithms that yield all the probabilities. The outputs generated by an algorithm and enclosed within brackets [-] are secret, e.g., sk_r is the secret output of $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{Commit}(pp, i)$. The following assumes that all statements are true for any λ, T and $pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$. ChronoCloak's properties are:

- *Correctness.* For every $\{m_0, m_1\} \leftarrow_{\$} \mathcal{M}^2$, and $i \in \{0, 1\}$ and for every honest challenge $\mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}')$ of an honest $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{Commit}(pp, i)$, $\mathsf{Open}(pp, [sk_r], \mathbf{c}, \mathbf{a}, \pi)$ always returns the intended m_i from an honest answer $(\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c})$ that Verify accepts with probability one.
- *Uniqueness.* ChronoCloak is unique only if, for every challenge $\mathbf{c} \in \mathcal{C}$ of $\mathbf{c}'$, there exists exactly one valid answer (i.e., $\mathbf{a}, \pi \leftarrow \mathsf{Solve}(pp, \mathbf{c})$), that Verify accepts, and this is the only answer through which Open returns $m \in \{m_0, m_1\}$, satisfying:

$$\Pr\left[\begin{array}{c} \mathbf{a}' \neq \mathbf{a} \\ \text{and} \\ \left(\begin{array}{c}\mathsf{Verify}(pp, \mathbf{a}', \pi') = \text{ACCEPT} \\ \text{or} \\ \mathsf{Open}(pp, sk'_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp\end{array}\right) \end{array} : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T),\ (\mathsf{pc}) \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ (\mathbf{a}, -) \leftarrow \mathsf{Solve}(pp, \mathbf{c}), \\ (\mathbf{a}', \pi', sk'_r) \leftarrow \mathsf{A}_3(pp, \mathsf{pc}, [sk_r], \mathbf{c}). \end{array}\right]$$

  is a negligible function of λ, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$. Although the answer $\mathbf{a}$ must be unique, the proof π does not.
- *Sequentiality.* ChronoCloak is sequential only if no adversary $\mathcal{A}$ can obtain the correct answer before an honest solver. Let $\mathsf{A}_{\text{short}}$ be an online algorithm that proposes $\mathbf{a}'$ in a shorter time than executing $\mathsf{Solve}(pp, \mathbf{c})$ (i.e., in less time than T). For any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_{\text{short}})$, we have

$$\Pr\left[\mathbf{a}' = \mathbf{a} : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T), \mathsf{pc} \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ \mathbf{a}' \leftarrow \mathsf{A}_{\text{short}}(pp, \mathsf{pc}, \mathbf{c}), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c}). \end{array}\right]$$

  is a negligible function of λ. This must also hold for both $\mathcal{A}_2 \in \mathcal{A}$ and $\mathcal{A}_2 \notin \mathcal{A}$.
- *Hiding.* ChronoCloak is hiding only if, for every challenge $\mathbf{c} \in \mathcal{C}$ of $\mathbf{c}'$, and answer $\mathbf{a} \leftarrow \mathsf{Solve}(pp, \mathbf{c})$, it holds that:

$$\left|\Pr\left[b' = b : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T),\ (\mathsf{pc}, [m_0, m_1]) \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{Commit}(pp, [i]),\ [b] \leftarrow_{\$} \{0, 1\}, \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', \left[\begin{array}{ll}[m_0, m_1] & \text{for } b = 0 \\ [m'_0, m'_1] \leftarrow_{\$} \mathcal{M}^2 & \text{for } b = 1\end{array}\right]), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c}),\ b' \leftarrow \mathsf{A}_4(pp, \mathsf{pc}, \mathbf{c}', \mathbf{c}, \mathbf{a}). \end{array}\right] - \frac{1}{2}\right|$$

  is a negligible function of λ, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_4)$.
- *Secrecy.* ChronoCloak is secretive only if there is no adversary, corrupting the receiver, inferring any extra information than inferred by an honest receiver. Hence, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3, \mathsf{A}_5, \mathsf{A}_6)$, we have

$$\Pr\left[\begin{array}{l} \hat{m}_{1-i} = m_{1-i}, \\ \text{and } \hat{m}_i = m_i \end{array} : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T),\ \mathsf{pc} \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ (\mathbf{a}', -, sk'_r) \leftarrow \mathsf{A}_3(pp, \mathsf{pc}, [sk_r], \mathbf{c}),\ \hat{m}_i \leftarrow \mathsf{A}_5(pp, \mathsf{pc}, [sk'_r], \mathbf{c}, \mathbf{a}') \\ \hat{m}_{1-i} \leftarrow \mathsf{A}_6(pp, \mathsf{pc}, [sk'_r], \mathbf{c}, \mathbf{a}', \hat{m}_i). \end{array}\right]$$

is a negligible function of λ.

– *Privacy.* ChronoCloak is private only if the $\mathbf{c}'_i$ of i is indistinguishable from $\mathbf{c}'_{i'}$ of i'. For any adversary $\mathcal{A}$, we have

$$\left| \mathsf{Pr}\left[\mathcal{A}(pp, \mathbf{c}') = i : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T), \\ [i] \leftarrow_{\$} \{0,1\}, \\ \mathbf{c}' \leftarrow_{\$} \mathsf{Commit}(pp, [i]) \end{array} \right] - 1/2 \right|$$

is a negligible function of λ.

The Ideal Functionality of ChronoCloak. Now, we present the ideal functionality $\mathsf{F_{CC}}$ that captures ChronoCloak features, shown in Fig. 5, which involves a series of interactions among participants. By submitting an Enrollment message

Enrollment: Upon receiving (Enroll, *sid*) from a party P_j, add P_j to $\mathcal{P}$; then send back (Enrolled, *sid*).
Commit: Upon receiving (Commit, *sid*, P_S, i) from P_R, add the tuple (P_R, P_S, *sid*, $m_0 = \varnothing$, $m_1 = \varnothing$, i, $T = \varnothing$, $\chi_0 = \varnothing$, $\chi_T = \varnothing$) to **InState**, where χ_j is the chronicle progress; and no further messages of this type is accepted. Output (Commit, *sid*, P_R) to P_S.
Challenge: Upon receiving (Challenge, *sid*, P_R, T, m_0, m_1) from P_S, if Commit is received, sample $(\mathbf{c}_j)_{j\in[T+1]} \leftarrow_{\$} \mathcal{C}^T$ s.t. $\nexists \mathbf{c}_j = \mathbf{c}_k \ \forall j \neq k$. Assign $\mathbf{ChronoSteps}[\mathbf{c}_j] = \mathbf{c}_{j+1} \ \forall j \in [T+1]$ and updates (P_S, P_R, *sid*, m_0, m_1, T, $\chi_0 = \mathbf{c}_1$, $\chi_T = \mathbf{c}_{T+1}$, $\pi_{\mathbf{list}} = [\varnothing]$) in **InState**. Lastly, broadcast (Challenge, *sid*, χ_0) to $\mathcal{P}$ and no further messages of this type is accepted.
Advance: Upon receiving (Step, *sid*, χ) from P_j, perform the following: (i) If $\mathbf{ChronoSteps}[\chi]$ is defined, add (P_j, Step, *sid*, χ, $\chi' := \mathbf{ChronoSteps}[\chi]$) to $\mathbf{L}_{in}$ and skip the next step. (ii) If $\mathbf{ChronoSteps}[\chi]$ is not defined, sample $\chi' \leftarrow_{\$} \mathcal{C}$, set $\mathbf{ChronoSteps}[\chi] = \chi'$ and add (P_j, Step, *sid*, χ, χ') to $\mathbf{L}_{in}$.
Prove: Upon receiving (Prove, *sid*, χ, χ') from P_j, add (P_j, Prove, *sid*, REJECT) to $\mathbf{L}_{in}$ if $(\chi, \chi') \neq (\chi_0, \chi_T)$, and skip next. A non-corrupt P_S: If there is no $(\mathsf{P}_j, \pi) \in \pi_{\mathbf{list}}$, append $(\mathsf{P}_j, \pi \leftarrow_{\$} \Pi)$ to $\pi_{\mathbf{list}}$, then add (P_j, Prove, *sid*, π) to $\mathbf{L}_{in}$. A corrupt P_S: Forward the query to $\tilde{\mathcal{A}}$ and wait to receive π'. If $\pi' \notin \Pi$, halt; otherwise add (P_j, Prove, *sid*, π') to $\mathbf{L}_{in}$.
Verify: Upon receiving (Verify, *sid*, χ, χ', π') from P_j, add (P_j, Verify, *sid*, REJECT) to $\mathbf{L}_{in}$ if $(\chi, \chi') \neq (\chi_0, \chi_T)$ or $(-, \pi') \notin \pi_{\mathbf{list}}$; otherwise add ($\mathsf{P}_j$, Verify, *sid*, ACCEPT) to $\mathbf{L}_{in}$.
Open: Upon receiving (Open, *sid*, χ, χ', π') from P_R, send P_R the response (Open, *sid*, REJECT) if $(\chi, \chi') \neq (\chi_0, \chi_T)$ or $(-, \pi') \notin \pi_{\mathbf{list}}$; if not, add ($\mathsf{P}_R$, Open, *sid*, m_i) to $\mathbf{L}_{in}$.
Tick: At every tick, set $\mathbf{L}_{out} \leftarrow \mathbf{L}_{in}$ and $\mathbf{L}_{in} \leftarrow \varnothing$.
Elicitate: Upon receiving (Elicitate, *sid*) from P_j, pop all entries in $\mathbf{L}_{out}$ with (P_j, $\cdots$) and send (Elicitate, *sid*, **resp**) to P_j, where **resp** denotes the popped entries. If (P_j, Prove, *sid*, π') $\in$ **resp**, send (P_j, (χ_T, π')) to P_R.

Fig. 5. The ideal functionality of ChronoCloak, $\mathsf{F_{CC}}$.

to $\mathsf{F_{CC}}$, participants are added to the Participant List ($\mathcal{P}$). A receiver P_R initiates interactions by submitting Commit to $\mathsf{F_{CC}}$ with a choice i. Upon receiving notice of P_R's query from $\mathsf{F_{CC}}$, the sender P_S responds by sending (m_0, m_1) via Challenge to $\mathsf{F_{CC}}$. $\mathsf{F_{CC}}$ broadcasts a challenge (χ_0) to all parties in $\mathcal{P}$. $\mathsf{P}_j \in \mathcal{P}$ prompts sequential Advance queries to $\mathsf{F_{CC}}$, obtaining $\{\chi_k\}_{k\in[T]}$ after all T-ticks, where to obtain χ_k at a tick, P_j must have χ_{k-1} from the previous tick. Note that, following each tick, all parties receive their respective responses from $\mathsf{F_{CC}}$ by calling Elicitate. Once T ticks have elapsed, P_j can obtain a proof π from $\mathsf{F_{CC}}$ by sending Prove with (χ_0, χ_T). Upon obtaining/receiving (χ_T, π') from $\mathsf{F_{CC}}$, P_R retrieves m_i by invoking Open on $\mathsf{F_{CC}}$, completing the ChronoCloak process.

5 Generic ChronoCloak

In this section, we present a generic ChronoCloak based on the functionalities $\mathsf{F_{RO}}$, $\mathsf{F_{OT}}$, and $\mathsf{F_{OU}}$. We then prove that our protocol securely realizes $\mathsf{F_{CC}}$, thereby proving that Definition 2 of ChronoCloak is simulatable implying the ideal functionality for $\mathsf{F_{CC}}$. In particular, we instantiate our protocol using Definition 2 of (Setup, Commit, Challenge, Solve, Verify, Open), and then we show that if our ChronoCloak is Commit-Challenge simulatable[1], then it is a UC-securely realizing $\mathsf{F_{CC}}$ in the random oracle mode. In light of this, we will use $\mathsf{F_{OT}}$, shown in Fig. 1, as a building block for Commit-Challenge. Additionally, our protocol achieves a time T delay through a long sequential function SeqEval with difficulty T. As discussed in Sect. 3, parties exclusively use $\mathsf{F_{OU}}$ to execute all functions, including the evaluation of SeqEval. Let Hash be a hash function, modeled by the random oracle $\mathsf{F_{RO}}$, defined as $\{0,1\}^* \to \{0,1\}^{2\lambda}$, where Hash takes arbitrary input and serializes it as a string of bits.

5.1 The Protocol

In Fig. 6, we present a ChronoCloak protocol $\Gamma_{\mathsf{F_{CC}}}$ based on Definition 2, which is defined by the parameters $(\mathcal{M}, \mathcal{S}, \mathcal{C}, \mathcal{Y} := \mathcal{C}, \Pi)$.

Discussion.

Correctness. For $\Gamma_{\mathsf{F_{CC}}}$ to be correct, we assume that SeqEval and $\mathsf{tr}_{sk} \in \mathcal{F}$ are commutative actions on $\mathcal{C}$'s elements for any $sk \in \mathcal{S}$ (i.e., for any element in the set $\mathcal{C}$, both sequences of actions—$\mathsf{tr}_{sk} \circ \mathsf{SeqEval}$ and $\mathsf{SeqEval} \circ \mathsf{tr}_{sk}$—are equivalent). Specifically, if $\hat{y} = \mathsf{tr}_{sk}(\mathsf{SeqEval}(x))$ and $\hat{y}' = \mathsf{SeqEval}(\mathsf{tr}_{sk}(x))$, then $\hat{y} = \hat{y}'$, $\forall x \in \mathcal{C}$ and $\forall sk \in \mathcal{S}$. The correctness requirement is feasible, as most efficient delay-based primitives with public verifiability rely on some algebraic structure satisfying the commutativity property. An example is the sequential function SeqEval introduced by Rivest et al. [41], which involves (T-times) repeated squaring in a group with an unknown order, G. In this context, $\mathsf{tr}_{sk}(\cdot)$ is then defined

[1] This means that a simulator can extract the receiver's (and the sender's) input during or after the Commit-Challenge phase, but before the solving process begins.

Initialization [Setup]. A trusted party $\mathsf{P}_{\mathrm{TTP}}$ generates a set of public parameters $pp \leftarrow_{\$} \mathsf{Setup}(\lambda, T)$ which includes (i) a description of $\mathsf{Solve.SeqEval} : \mathcal{C} \rightarrow \mathcal{C}^a$ to be the T-long function with a public challenge and answer (i.e., $x, y \in \mathcal{C}$ such that $y \leftarrow \mathsf{SeqEval}(x)$), and (ii) a description of a large ensemble of short trapdoors $\mathcal{F}$ given a secret space $\mathcal{S}$ (i.e., $\mathsf{tr}_{sk} \leftarrow \mathcal{F}$ given $sk \leftarrow_{\$} \mathcal{S}$). $\mathsf{P}_{\mathrm{TTP}}$ sends (Setup, $\mathsf{P}_{\mathrm{TTP}}$, pp) to all parties.
Preparation [Enrollment]. Initially, P_S and P_R share a common input pp and $sid \in \{0,1\}^*$. Using a private input $i \in \{0,1\}$, the receiver (P_R) decides which message to obtain. In response to a receiver's request, the sender (P_S) prepares the relevant message set $(m_0, m_1) \in \mathcal{M}^2$.
Message 1 [Commit]. Instantiated by F_{OT}, P_R sends (Commit, sid, P_S, i) to F_{OT}.
Message 2 [Challenge]. Upon receiving (Commit, sid, P_R) from F_{OT}, P_S performs the following

- computes $\hat{x} \leftarrow \mathsf{tr}_{sk_c}(x)$ and $\hat{y} \leftarrow \mathsf{tr}_{sk_c}(y)$, where $\mathsf{tr}_{sk_c} \leftarrow \mathcal{F}$ for $sk_c \leftarrow_{\$} \mathcal{S}$.
- sends (Query, sid, $\hat{x} \,\|\, \hat{y}$) to F_{RO}, and waits to receive (Query, sid, k). Then it sets $k^{\mathrm{MSB}} \,\|\, k^{\mathrm{LSB}} = k$, where k^{MSB} and $k^{\mathrm{LSB}} \in \{0,1\}^{\lambda}$.
- computes $\hat{m_0} = m_0 \oplus k^{\mathrm{MSB}}$, $\hat{m_1} = m_1 \oplus k^{\mathrm{MSB}}$, and $\mathsf{sig} := sk_c \oplus k^{\mathrm{LSB}}$.
- sends (Query, sid, $\mathsf{sig} \,\|\, sk_c \,\|\, k^{\mathrm{LSB}}$) to F_{RO}, and waits to receive (Query, sid, cm_{sig}).

P_S sends ($\mathsf{Challenge}$, sid, P_R, $\hat{m_0}, \hat{m_1}$) to F_{OT}. If ($\mathsf{Challenge}$, sid, P_R) is received, P_S sends ($\mathsf{Challenge}$, sid, P_S, $\mathbf{c} := (\hat{x}, \mathsf{sig}, cm_{\mathsf{sig}})$) to P_R.

Message 3 [Solve]. Upon receiving ($\mathsf{Challenge}$, sid, P_S, $\hat{m_i}$) from F_{OT} and ($\mathsf{Challenge}$, sid, P_S, $\mathbf{c}$) from P_S, P_R sends ($\mathsf{Evaluate}$, sid, Solve, $\hat{in} := (pp, \mathbf{c}.\hat{x})$) to F_{OU} and awaits to receive ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_{\mathsf{Solve}}$, T). To obtain ($\mathbf{a}' := \hat{y}', \pi \in \Pi$), P_R sends T consecutive ($\mathsf{Elicitate}$, sid) queries to the F_{OU}, stopping upon receiving the T^{th} response ($\mathsf{Elicitate}$, sid, $\mathbf{resp}$) with a non-empty $\mathbf{resp} \neq \varnothing$.
Output [Open]. Upon obtaining $(\hat{y}', \pi)$, P_R performs the following

- checks that $\mathsf{Verify}(pp, \mathbf{c}.\hat{x}, (\hat{y}', \pi)) = \textsc{accept}$; if no, it outputs $\perp$.
- sends (Query, sid, $\hat{x} \,\|\, \hat{y}'$) to F_{RO}, and waits to receive (Query, sid, k). Then it sets $k^{\mathrm{MSB}} \,\|\, k^{\mathrm{LSB}} = k$.
- computes $sk_c := \mathbf{c}.\mathsf{sig} \oplus k^{\mathrm{LSB}}$; then it sends ($\mathsf{Query}$, sid, $\mathbf{c}.\mathsf{sig} \,\|\, sk_c \,\|\, k^{\mathrm{LSB}}$) to F_{RO}, and waits to receive (Query, sid, cm'_{sig}).
- checks that $cm'_{\mathsf{sig}} = \mathbf{c}.cm_{\mathsf{sig}}$, $\hat{x} = \mathsf{tr}_{sk_c}(x)$, and $\hat{y} = \mathsf{tr}_{sk_c}(y)$; if no, it outputs $\perp$; otherwise, it continues next.

Finally, P_R computes $m_i = \hat{m_i} \oplus k^{\mathrm{MSB}}$

[a] We use this notation (i.e., $\mathsf{Solve.SeqEval}$) to indicate the function $\mathsf{SeqEval}$ being a part of Solve algorithm.

Fig. 6. Proposed ChronoCloak protocol $\Gamma_{\mathsf{F_{CC}}}$ implementing $\mathsf{F_{CC}}$.

as $\mathsf{tr}_{sk}(x) = x^{sk}$. For all $x \in G$, the functions $\mathsf{SeqEval}$ and tr_{sk} commute as follows: $\mathsf{SeqEval} \circ \mathsf{tr}_{sk}(x) = (x^{sk})^{2^T} = \mathsf{tr}_{sk}(x^{2^T}) = \mathsf{tr}_{sk} \circ \mathsf{SeqEval}(x)$, which satisfies $\hat{y} = \mathsf{tr}_{sk}(y)$. Another example can be found in the work of [46].

Alternative Approach. When designing a ChronoCloak protocol, it may seem intuitive to time-lock each secret message with a challenge and then use $\mathsf{F_{OT}}$ to transmit either of these challenges. This approach, however, may leak the receiver's choice in two instances: the first instance is in the public-solving model, in which the receiver reveals the transmitted challenge to the public solver. One could argue that the receiver may obfuscate the transmitted challenge to conceal their choice before sending it to the solver; however, this obfuscation method has several problems, such as the difficulty of disputing an improperly constructed challenge (or commitment) without the receiver revealing their concealed secret and thus their choice. The second instance results from a selective fault attack (by the sender) in which one of the messages contains a maliciously corrupted challenge.

Furthermore, our construction is more efficient. Compared to the alternative approach, which requires the creation of two challenges, their commitments (i.e., two commitments), one OT operation, and one concealing operation (for a public solver), while our construction only requires the creation of one challenge, its commitment, and one OT operation.

Real-World Example. The constructed protocol of ChronoCloak must maintain being a "function" (i.e., being unique), where each challenge has only one solution that allows message extraction, and which the verifier accepts (i.e., it is infeasible to find another answer that Verify outputs accepts or that Open outputs anything but $\perp$). Consider a real-world example: If a sender transmits a valid challenge $\mathbf{c}$ (that has an answer $\mathbf{a}$), and the receiver finds an alternative answer $\mathbf{a}'$ (with its proof) that also passes verification, the receiver can then show that $\mathbf{c}$ is not properly constructed given $\mathbf{a}'$. This can affect applications where the receiver has to compensate for received content.

Scalability. Our ChronoCloak protocol scales well for multiple receivers, allowing a sender to transmit content to many receivers with a single challenge, though requiring one OT operation for each receiver. The protocol scales well because the solution to the challenge ensures that all receivers obtain their chosen content *obliviously*. However, this work does not extend effectively to the scenario of multiple senders and receivers, where each sender needs to generate its own challenge and a corresponding solution, thereby affecting scalability. The solution to this limitation was left for future work.

5.2 Security

We will now provide a proof that $\Gamma_{\mathsf{F_{CC}}}$ securely implements $\mathsf{F_{CC}}$.

Theorem 1. *The protocol $\Gamma_{\mathsf{F_{CC}}}$ is securely UC-implementing $\mathsf{F_{CC}}$ in ($\mathsf{F_{RO}}$, $\mathsf{F_{OT}}$, $\mathsf{F_{OU}}$)-hybrid model under a malicious adversary (A) with static corruption, and for any PPT environment Env.*

Proof. To prove the theorem, we construct the simulators $\tilde{\mathcal{A}}'_S$ and $\tilde{\mathcal{A}}'_R$ (see Figs. 7 and 8). Env produces inputs sent to the appropriate parties (i.e., $\mathsf{P}_S \xleftarrow{send} (m_0, m_1), \mathsf{P}_R \xleftarrow{send} i, \mathsf{A} \xleftarrow{send} \mathsf{aux}$). The simulator provides *the malicious adversary* with the matching inputs. $\tilde{\mathcal{A}}'_S$ (resp. $\tilde{\mathcal{A}}'_R$) simulates $\Gamma_{\mathsf{F_{CC}}}$ in the presence of a copy of A_S (resp. A_R), corrupting the sender (resp. receiver, and solver). The simulations emulate the random oracle $\mathsf{F_{RO}}$ with empty state. Hence, it responds to any query from/to $\mathsf{F_{RO}}$ and keeps track of all queries and responses.

Security with Corrupted Sender and Honest Receiver. We present $\tilde{\mathcal{A}}'_S$ who passes the inputs $([m_0, m_1], \mathsf{aux})$ from Env to A_S.

1. *Initialization and Preparation*: On Env's inputs (λ, T), $\tilde{\mathcal{A}}'_S$ simulates Setup honestly. Then, A_S is invoked with inputs (m_0, m_1, aux) from Env.
2. *Message 1 [Commit]*: As an honest receiver, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Commit}, sid, \mathsf{P}_R, i' = 0)$ to $\mathsf{F_{OT}}$ with a dummy input (e.g., $i' = 0$).
3. *Message 2 [Challenge]*: Upon receiving $(\mathsf{Challenge}, sid, \mathsf{A}_S, \hat{m}_i)$ from $\mathsf{F_{OT}}$ and $(\mathsf{Challenge}, sid, \mathsf{A}_S, \mathbf{c} = (\hat{x}, \mathsf{sig}, cm_{\mathsf{sig}}))$ from A_S, $\tilde{\mathcal{A}}'_S$ observes all queries (i.e., $\mathsf{inString} \in \mathsf{F_{RO}}.\mathbf{H}$) to $\mathsf{F_{RO}}$. Accordingly, $\tilde{\mathcal{A}}'_S$ identifies a record in $\mathsf{F_{RO}}.\mathbf{H}$ with a prefix $(str_1, \hat{\mathbf{c}}.\hat{x} \parallel str_2) \in \mathsf{F_{RO}}.\mathbf{H}$, which allows it to obtain $k^{\mathrm{MSB}} \parallel k^{\mathrm{LSB}} = k \leftarrow str_1$, the answer $\hat{y} \leftarrow str_2$, and the secret $sk_c := \mathbf{c}.\mathsf{sig} \oplus k^{\mathrm{LSB}}$ (Note that there might be multiple records with the same prefix, i.e., $\hat{\mathbf{c}}.\hat{x} \parallel str_2$). $\tilde{\mathcal{A}}'_S$ also identifies a record of form $(str_1, \mathbf{c}.\mathsf{sig} \parallel sk_c \parallel k^{\mathrm{LSB}}) \in \mathsf{F_{RO}}.\mathbf{H}$, which allows it to obtain $cm'_{\mathsf{sig}} \leftarrow str_1$, if there is no record of such, it sets $cm'_{\mathsf{sig}} = \perp$. Following that, it checks for $\hat{y}$ as follows: $cm'_{\mathsf{sig}} \stackrel{?}{=} \mathbf{c}.cm_{\mathsf{sig}}$, $\hat{x} \stackrel{?}{=} \mathsf{tr}_{sk_c}(x)$, and $\hat{y} \stackrel{?}{=} \mathsf{tr}_{sk_c}(y)$. If the validation passes, $\tilde{\mathcal{A}}'_S$ computes $m_0 = \mathbf{c}.\hat{m}_0 \oplus k^{\mathrm{MSB}}$, $m_1 = \mathbf{c}.\hat{m}_1 \oplus k^{\mathrm{MSB}}$; otherwise, $\tilde{\mathcal{A}}'_S$ sets $(m_0, m_1) \leftarrow_{\$} \mathcal{M}^2$.
4. *Message 3 [Solve]*: After above, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Challenge}, sid, \mathsf{P}_R, T, m_0, m_1)$ to $\mathsf{F_{CC}}$, and waits for $(\mathsf{Challenge}, sid, \chi_0)$. Then, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{Solve}} \leftarrow \tilde{\mathcal{A}}'_R(\mathsf{Solve}), in \leftarrow \tilde{\mathcal{A}}'_R(pp, \mathbf{c}.\hat{x}))$ to $\mathsf{F_{OU}}$, and receives $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{Solve}}, \tau)$. In the event that A_S did not query either or both of $(\mathsf{Query}, sid, \hat{x} \parallel str_1)$ and $(\mathsf{Query}, sid, \mathsf{sig} \parallel sk_c \parallel k^{\mathrm{LSB}})$, $\tilde{\mathcal{A}}'_S$ tracks all queries to $\mathsf{F_{RO}}$, and if a query for an answer passes the validation in Step 3, it responses in accordance to the messages submitted to $\mathsf{F_{CC}}$ (i.e., m_0, m_1).
5. *Output [Open]*: After τ ticks, $\tilde{\mathcal{A}}'_S$ responds to all Prove query from $\mathsf{F_{CC}}$ with $\perp$
   - if $\mathbf{a} = \hat{y}$ obtained from Step 3 is not equal to $\hat{y}'$ obtained from $\mathsf{F_{OU}}$ (and skip next);
   - and if the challenge $\mathbf{c}$ is not well formed, i.e., $cm'_{\mathsf{sig}} \neq \mathbf{c}.cm_{\mathsf{sig}}$, $\hat{x} \neq \mathsf{tr}_{sk_c}(x)$, or $\hat{y} \neq \mathsf{tr}_{sk_c}(y)$ for sk_c obtained using $\hat{y}$ (and skip next);
   - otherwise it responds with $\pi \in \Pi$.

   Lastly, once P_R, in the ideal execution, receives m_i, $\tilde{\mathcal{A}}'_S$ returns A_S's output to Env.

Fig. 7. A simulation $\tilde{\mathcal{A}}'_S$ for $\Gamma_{\mathsf{F_{CC}}}$.

Below, we show that $\tilde{\mathcal{A}}'_S$'s output is indistinguishable from the real execution and aligns with the ideal execution. $\tilde{\mathcal{A}}'_S$ operates honestly within the simulation

by generating pp and acting as a trusted receiver, rendering it indistinguishable from a real execution. Furthermore, in Step 2, $\tilde{\mathcal{A}}'_S$ does not know P_R's input (in the ideal case), so it uses fixed choice, i.e., $i' = 0$. Despite this, Env cannot distinguish whether i' matches P_R's input or not. This is due to ChronoCloak's 'privacy' assumption, which holds given the ideal functionality F_{OT}. As such, the two executions remain identical.

Additionally, to participate in the ideal execution, $\tilde{\mathcal{A}}'_S$ must send m_0 and m_1 to the F_{CC} at an early stage (i.e., before obtaining $\hat{y}$). However, since $\tilde{\mathcal{A}}'_S$ does not possess both secret messages, m_0 and m_1, it cannot consistently participate in the ideal execution as a real execution would. Fortunately, in Step 3, $\tilde{\mathcal{A}}'_S$ can extract and forward both secret messages m_0 and m_1 using sig, cm_{sig} and through F_{RO}. If it cannot extract both messages, $\tilde{\mathcal{A}}'_S$ will select a random message(s) of the same length (i.e., representing the case in which the adversary did not inquire F_{RO}). This indicates that A_S sends a random string(s) (i.e., $\hat{m}_0$, and $\hat{m}_1$), which implies sending a message(s) that are unknown to A_S (since A_S did not acquire the corresponding key, k^{MSB}). Later on, if A_S makes additional query(s) for the missing key, $\tilde{\mathcal{A}}'_S$ returns a suitable key that corresponds to the message sent to F_{CC} (in Step 4). However, the probability of A_S forming a valid challenge that passes in Step 5 is negligible since it has to predict the output of the query to F_{RO} in such a way that $k^{\mathrm{MSB}} = \mathsf{sig} \oplus sk_c$ (same for cm_{sig}). In this case, the simulator sends a null proof message. Accordingly, the $\tilde{\mathcal{A}}'_S$ can participate in ideal execution with interaction consistent with input from Env and the adversary's strategy. As a result, both executions are indistinguishable.

Security with Honest Sender and Corrupted Receiver. We present $\tilde{\mathcal{A}}'_R$ who passes the inputs $([i], \mathsf{aux})$ from Env to A_R.

Below, we show that the view generated by $\tilde{\mathcal{A}}'_R$ cannot be distinguished from the real execution and is consistent with the ideal execution. While in the simulation, $\tilde{\mathcal{A}}'_R$ behaves primarily as an honest sender, it deviates from honest execution in Step 3 (as it does not have P_S's inputs m_0 and m_1, in the ideal case). In more detail, $\tilde{\mathcal{A}}'_R$ sends to F_{OT} bogus challenge (i.e., $(\hat{m}_0, \hat{m}_1) \leftarrow_{\$} \mathcal{M}^2$, $\mathsf{sig} \leftarrow_{\$} \{0,1\}^{\lambda}$, and $cm_{\mathsf{sig}} \leftarrow_{\$} \mathsf{F}_{\mathsf{RO}}.\mathcal{H}$) hoping to receive $m_{i'}$ from F_{CC} later on (i.e., after τ ticks). For this reason, $\tilde{\mathcal{A}}'_R$ must first extract A_R's choice, i.e., i', which it can with the help of F_{OT} (see Step 2). So far, $\tilde{\mathcal{A}}'_R$ has sent (to F_{OT}) two random messages, which is equivalent to sending true messages with an unknown key (i.e., $\hat{m}_{i'} = m_{i'} \oplus k^{\mathrm{MSB}}$ for $k^{\mathrm{MSB}} \leftarrow_{\$} \{0,1\}^{\lambda}$). Env hence distinguishes between both executions if it can determine whether $\hat{m}_{i'}$ generated at random or through $(k^{\mathrm{MSB}} \parallel \cdots) \leftarrow \mathsf{Hash}(\hat{x} \parallel \hat{y})$. This event is negligible since Hash is assumed to be modeled as a random oracle; thus, the probability of distinguishing between executions is negligible without the knowledge of $\hat{y}$.

Further, $\tilde{\mathcal{A}}'_R$ in Step 4 advances step by step at each tick to retrieve χ_T, enabling it to obtain $m_{i'}$ from F_{CC}. However, $\tilde{\mathcal{A}}'_R$ differs from the real execution in that it aborts when A_R sends a query of form $(\mathsf{Query}, sid, \hat{x} \parallel \hat{y})$ (to F_{RO}) before $\tilde{\mathcal{A}}'_R$. The probability of this abort is equal to the probability of A_R breaking the ChronoCloak's 'sequentiality' assumption of Solve (i.e., obtaining the answer $\mathbf{a}$ before time T). After τ ticks, $\tilde{\mathcal{A}}'_R$ can only know one of the two

1. *Initialization and Preparation*: The simulator obtains (λ, T) from the environment as a common public input. Then, A_R is invoked with inputs (i, aux) from Env.
2. *Message 1 [Commit]*: Upon receiving $(\mathsf{Commit}, \mathsf{A}_R, sid)$ from F_{OT}, $\tilde{\mathcal{A}}'_R$ receives A_R's choice i' (which may differ from i) from A_R's messages to F_{OT}.
3. *Message 2 [Challenge]*: After above, $\tilde{\mathcal{A}}'_S$ samples two bogus messages $(\hat{m}_0, \hat{m}_1) \leftarrow_{\$} \mathcal{M}^2$. Then, it computes $\mathbf{c} := (\hat{x} \leftarrow \mathsf{tr}_{sk_c}(x), \mathsf{sig} \leftarrow_{\$} \{0,1\}^{\lambda}, cm_{\mathsf{sig}} \leftarrow_{\$} \mathsf{F}_{\mathsf{RO}}.\mathcal{H})$ and $\hat{y} \leftarrow \mathsf{tr}_{sk_c}(y)$, where $\mathsf{tr}_{sk_c} \leftarrow \mathcal{F}$ for $sk_c \leftarrow_{\$} \mathcal{S}$. Finally, it sends $(\mathsf{Challenge}, sid, \mathsf{A}_R, \hat{m}_0, \hat{m}_1)$ to F_{OT}. If $(\mathsf{Challenge}, sid, \mathsf{A}_R)$ is received, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Challenge}, sid, \mathsf{P}_S, \mathbf{c} := (\hat{x}, \mathsf{sig}, cm_{\mathsf{sig}}))$ to A_R.
   Simultaneously, $\tilde{\mathcal{A}}'_R$ sends $(\mathsf{Commit}, sid, \mathsf{P}_S, i')$ to F_{CC}. After that, it awaits the F_{CC}'s $(\mathsf{Challenge}, sid, \chi_0)$ messages. Lastly. $\tilde{\mathcal{A}}'_R$ sends $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{SeqEval}} \leftarrow \tilde{\mathcal{A}}'_R(\mathsf{Solve.SeqEval}), in \leftarrow \tilde{\mathcal{A}}'_R(\mathbf{c}.\hat{x}))$ to F_{OU}, and receives $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{SeqEval}}, \tau)$.
4. *Message 3 [Solve]*: At every tick, $\tilde{\mathcal{A}}'_R$ obtains the current sequential step χ' by sending $(\mathsf{Elicitate}, sid)$ to F_{CC}. Then, $\tilde{\mathcal{A}}'_R$ updates $\tau \leftarrow \tau - 1$ and sends $(\mathsf{Step}, sid, \chi')$ to F_{CC}. $\tilde{\mathcal{A}}'_R$ outputs ABORT, if $\tau > 0$ and $\tilde{\mathcal{A}}'_R$ observes a query sent to F_{RO} by A_R of form $(\mathsf{Query}, sid, \hat{x} \,\|\, \hat{y})$.
5. *Output [Open]*: When $\tau = 0$, $\tilde{\mathcal{A}}'_R$ obtains $m_{i'}$ from F_{CC} by sending Prove and Open calls. Upon receiving $(\mathsf{Query}, sid, \hat{x} \,\|\, \hat{y})$ from A_R, $\tilde{\mathcal{A}}'_R$ computes $k^{\mathrm{MSB}} = m_{i'} \oplus \hat{m}_{i'}$ and $k^{\mathrm{LSB}} = sk_c \oplus \mathsf{sig}$, and sends $(\mathsf{Query}, sid, k^{\mathrm{MSB}} \,\|\, k^{\mathrm{LSB}})$ to A_R. Also, for any query of form $(\mathsf{Query}, sid, \mathsf{sig} \,\|\, sk_c \,\|\, k^{\mathrm{LSB}})$, $\tilde{\mathcal{A}}'_R$ responds by $(\mathsf{Query}, sid, cm_{\mathsf{sig}})$.

Fig. 8. A simulation $\tilde{\mathcal{A}}'_R$ for $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$.

messages (i.e., only $m_{i'}$), whereas Env knows both (i.e., $m_{i'}$ and $m_{1-i'}$). Env would, however, require to obtain the other message $\hat{m}_{1-i'}$ sent through F_{OT} (by $\tilde{\mathcal{A}}'_R$) to distinguish $m_{1-i'}$ from $\overline{m}_{1-i'} := \hat{m}_{1-i'} \oplus k^{MSB}$. Therefore, this is infeasible due to ChronoCloak's 'secrecy' assumption, which preserved through the functionality of F_{OT}. In Step 5, $\tilde{\mathcal{A}}'_R$ can answer A_R's query to F_{RO} with the appropriate key $k^{\mathrm{MSB}} \,\|\, k^{\mathrm{LSB}} = k$ in accordance with $m_{i'}$. Yet, both executions will become distinguishable in case A_R obtains an alternative valid answer $\hat{y}'$ $(\neq \hat{y})$ accepted by Verify, in which $\tilde{\mathcal{A}}'_R$'s challenge $\mathbf{c}$ is quite unlikely to be well-formed given $\hat{y}'$ under the Hash assumption. This contradicts ChronoCloak's 'uniqueness' assumption and is therefore infeasible. As a result, $\tilde{\mathcal{A}}'_R$ can simulate the protocol with interactions that are consistent with inputs from the environment and the strategy of the adversary, whereby the two executions are indistinguishable.

6 Conclusion

In this article, we have presented a new construct called ChronoCloak, intended to address the issue of premature exposure of sender's data and privacy-preserving of receiver's interaction. In the context of oblivious digital dissem-

ination, ChronoCloak has the potential to be a viable integrated solution to mitigate premature disclosure. As opposed to its existing counterparts, the proposed ChronoCloak is based on an integrated model and does not require a time server. Essentially, ChronoCloak enables a sender to transmit secrets through a puzzle that must be solved via lengthy computation. By solving the puzzle, only a subset of the secrets is revealed to the receiver, which is oblivious to the sender. Moreover, the receiver may outsource the computation (with public verification), but only the intended recipient is able to retrieve a subset of secrets using the solution to the puzzle. In this work, we define an ideal functionality for ChronoCloak and present a protocol to implement this functionality in the random oracle model by integrating an ideal OT functionality with a time lock-like function.

Acknowledgments. We sincerely thank the anonymous reviewers for their valuable comments and thoughtful suggestions.

References

1. Abusalah, H., Kamath, C., Klein, K., Pietrzak, K., Walter, M.: Reversible proofs of sequential work. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 277–291. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_10
2. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 411–439. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_14
3. Arapinis, M., Lamprou, N., Zacharias, T.: Astrolabous: a universally composable time-lock encryption scheme. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021. LNCS, vol. 13091, pp. 398–426. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92075-3_14
4. Badrinarayanan, S., Masny, D., Mukherjee, P.: Efficient and tight oblivious transfer from PKE with tight multi-user security. In: Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, LNCS, vol. 13269, pp. 626–642. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-09234-3_31
5. Badrinarayanan, S., Masny, D., Mukherjee, P., Patranabis, S., Raghuraman, S., Sarkar, P.: Round-optimal oblivious transfer and MPC from computational CSIDH. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography - PKC 2023. LNCS, vol. 13940, pp. 376–405. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-31368-4_14
6. Barreto, P., Oliveira, G., Benits, W.: Supersingular isogeny oblivious transfer. IACR Cryptol. ePrint Arch. p. 459 (2018), https://eprint.iacr.org/2018/459
7. Barreto, P.S.L.M., David, B., Dowsley, R., Morozov, K., Nascimento, A.C.A.: A framework for efficient adaptively secure composable oblivious transfer in the ROM. CoRR **abs/1710.08256** (2017). http://arxiv.org/abs/1710.08256
8. Baum, C., David, B., Dowsley, R., Kishore, R., Nielsen, J.B., Oechsner, S.: CRAFT: composable randomness beacons and output-independent abort MPC from time. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography - PKC 2023. LNCS, vol. 13940, pp. 439–470. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-31368-4_16

9. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: TARDIS: a foundation of time-lock puzzles in UC. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 429–459. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_15
10. Beullens, W., Kleinjung, T., Vercauteren, F.: CSI-FiSh: efficient isogeny based signatures through class group computations. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 227–247. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_9
11. Branco, P., Döttling, N., Srinivasan, A.: A framework for statistically sender private OT with optimal rate. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. Lecture Notes in Computer Science, vol. 14081, pp. 548–576. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38557-5_18
12. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science, pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888
13. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptol. **19**(2), 135–167 (2006). https://doi.org/10.1007/s00145-005-0419-9
14. Canetti, R., Sarkar, P., Wang, X.: Efficient and round-optimal oblivious transfer and commitment with adaptive security. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12493, pp. 277–308. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64840-4_10
15. Castryck, W., Decru, T.: An efficient key recovery attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology - EUROCRYPT 2023. LNCS, vol. 14008, pp. 423–447. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30589-4_15
16. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology - ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
17. Chávez-Saab, J., Rodríguez-Henríquez, F., Tibouchi, M.: Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In: Selected Areas in Cryptography - 28th International Conference, SAC 2021. LNCS, vol. 13203, pp. 441–460. Springer (2021). https://doi.org/10.1007/978-3-030-99277-4_21
18. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23–25 October 1995, pp. 41–50. IEEE Computer Society (1995). https://doi.org/10.1109/SFCS.1995.492461
19. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: Lauter, K., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 40–58. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22174-8_3
20. Cohen, B., Pietrzak, K.: Simple proofs of sequential work. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 451–467. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_15
21. Di Crescenzo, G., Ostrovsky, R., Rajagopalan, S.: Conditional oblivious transfer and timed-release encryption. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 74–89. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_6
22. David, B., Dowsley, R., Nascimento, A.C.A.: Universally composable oblivious transfer based on a variant of LPN. In: Gritzalis, D., Kiayias, A., Askoxylakis, I.

(eds.) CANS 2014. LNCS, vol. 8813, pp. 143–158. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12280-9_10
23. Döttling, N., Garg, S., Hajiabadi, M., Masny, D., Wichs, D.: Two-round oblivious transfer from CDH or LPN. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 768–797. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_26
24. Eldefrawy, K., Jakkamsetti, S., Terner, B., Yung, M.: Standard model time-lock puzzles: defining security and constructing via composition. IACR Cryptol. ePrint Arch, p. 439 (2023). https://eprint.iacr.org/2023/439
25. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12107, pp. 125–154. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_5
26. Freitag, C., Komargodski, I., Pass, R., Sirkin, N.: Non-malleable time-lock puzzles and applications. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13044, pp. 447–479. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90456-2_15
27. Genç, Z.A., Iovino, V., Rial, A.: The simplest protocol for oblivious transfer revisited. Inf. Process. Lett. **161**, 105975 (2020). https://doi.org/10.1016/J.IPL.2020.105975
28. Hazay, C., Lindell, Y.: Efficient Secure Two-Party Protocols: Techniques and Constructions, 1st edn. Springer-Verlag, Berlin, Heidelberg (2010)
29. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32
30. Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12552, pp. 390–413. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64381-2_14
31. Lai, Y.-F., Galbraith, S.D., Delpech de Saint Guilhem, C.: Compact, efficient and UC-secure isogeny-based oblivious transfer. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 213–241. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_8
32. Lai, Y.F., Galbraith, S.D., de Saint Guilhem, C.D.: Compact, efficient and uc-secure isogeny-based oblivious transfer. Cryptology ePrint Archive, Paper 2020/1012 (2020). https://eprint.iacr.org/2020/1012
33. Lenstra, A.K., Wesolowski, B.: Trustworthy public randomness with sloth, unicorn, and TRX. Int. J. Appl. Cryptogr. **3**(4), 330–343 (2017). https://doi.org/10.1504/IJACT.2017.10010315
34. Li, B., Micciancio, D.: Equational security proofs of oblivious transfer protocols. In: Abdalla, M., Dahab, R. (eds.) PKC 2018. LNCS, vol. 10769, pp. 527–553. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-76578-5_18
35. Loe, A.F., Medley, L., O'Connell, C., Quaglia, E.A.: A practical verifiable delay function and delay encryption scheme. IACR Cryptol. ePrint Arch. p. 1293 (2021). https://eprint.iacr.org/2021/1293
36. Ma, X., Xu, L., Zhang, F.: Oblivious transfer with timed-release receiver privacy. J. Syst. Softw. **84**(3), 460–464 (2011). https://doi.org/10.1016/j.jss.2010.11.886
37. Mahmoody, M., Moran, T., Vadhan, S.P.: Publicly verifiable proofs of sequential work. In: Kleinberg, R.D. (ed.) Innovations in Theoretical Computer Science, ITCS 2013, Berkeley, CA, USA, 9-12 January 2013, pp. 373–388. ACM (2013). https://doi.org/10.1145/2422436.2422479

38. Micciancio, D., Sorrell, J.: Simpler statistically sender private oblivious transfer from ideals of cyclotomic integers. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020. LNCS, vol. 12492, pp. 381–407. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64834-3_13
39. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (ed.) 10th Innovations in Theoretical Computer Science Conference, ITCS 2019, 10–12 January 2019, San Diego, California, USA. LIPIcs, vol. 124, pp. 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ITCS.2019.60
40. Rabin, M.O.: How to exchange secrets with oblivious transfer. Tech. Report. TR-81; Aiken Computation Lab, Harvard University: Cambridge, MA, USA (1981)
41. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical Report, Massachusetts Institute of Technology, USA (1996)
42. de Saint Guilhem, C.D., Orsini, E., Petit, C., Smart, N.P.: Semi-commutative masking: a framework for isogeny-based protocols, with an application to fully secure two-round isogeny-based OT. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 2020. LNCS, vol. 12579, pp. 235–258. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-65411-5_12
43. Vitse, V.: Simple oblivious transfer protocols compatible with supersingular isogenies. In: Buchmann, J., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2019. LNCS, vol. 11627, pp. 56–78. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-23696-0_4
44. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13
45. Xu, L., Zhang, F., Tang, S.: Timed-release oblivious transfer. Secur. Commun. Netw. **7**(7), 1138–1149 (2014). https://doi.org/10.1002/sec.845
46. Zawia, A., Hasan, M.A.: A new class of trapdoor verifiable delay functions. In: Jourdan, G.V., Mounier, L., Adams, C., Sèdes, F., Garcia-Alfaro, J. (eds.) Foundations and Practice of Security, pp. 71–87. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-30122-3_5