On the Security of Time-Locked Puzzles and Timed Commitments

Jonathan Katz[1], Julian Loss[2], and Jiayu Xu[1]

[1] George Mason University
[2] University of Maryland

Abstract. Time-locked puzzles—problems whose solution requires some amount of (inherently) sequential effort—have seen a recent increase in popularity (e.g., in the context of verifiable delay functions). Most constructions rely on the conjecture that, given a random x, computing $x^{2^T} \bmod N$ requires at least T (sequential) steps. We study the security of time-locked primitives from two perspectives:

1. We give the first hardness result about the sequential squaring conjecture. Namely, we show that even in (a quantitative version of) the algebraic group model, any speed up of sequential squaring is as hard as factoring N.
2. We then focus on *timed commitments*, one of the most important primitives that can be obtained from time-locked puzzles. We extend existing security definitions to settings that may arise when using timed commitments in higher-level protocols. We then give the first construction of *non-malleable* timed commitments. As a building block of independent interest, we also define (and give constructions for) a new primitive called *time-released public-key encryption*.

1 Introduction

Time-locked puzzles, introduced by Rivest, Shamir, and Wagner [26], refer to a fascinating type of computational problem that requires a certain amount of sequential effort to solve. Using time-locked puzzles, it is possible to "encrypt a message into the future" such that it remains computationally hidden for some time T, but can be recovered once this time has passed. Various primitives can be built from time-locked puzzles, including verifiable delay functions [4, 5, 25, 28], zero-knowledge proofs [11], non-malleable commitments [16], and timed commitments [6]. These, in turn, have applications such as fair coin tossing, e-voting, auctions, and contract signing [6, 20]. In spite of the increasing popularity of time-locked puzzles and timed commitments, their formal analysis has received only limited attention so far (to the best of our knowledge)—with few exceptions [6, 20]. In this work, we seek to improve confidence in both of these primitives, by (1) providing the first formal evidence supporting the hardness of the most widely used time-locked puzzle [26], and (2) giving new, stronger security definitions (and constructions) for timed commitments and related primitives.

Hardness in the (strong) AGM. The hardness assumption underlying the most popular time-locked puzzle [26] is that, given a random element x in the quadratic residue[1] group $\mathbb{QR}_N$ (where N is the product of two large safe primes), it is hard to compute $x^{2^T} \bmod N$ in fewer than T steps. We study this assumption in a new, strengthened version of the algebraic group model (AGM) [12] called the *strong AGM (SAGM)* that lies in between the generic group model (GGM) [21, 27] and the AGM. Roughly, an algorithm $\mathcal{A}$ in the AGM is constrained as follows: for any group element x that $\mathcal{A}$ outputs, $\mathcal{A}$ must also output algebraic coefficients showing how x was computed from $\mathcal{A}$'s

[1] The problem was originally stated over the ring $\mathbb{Z}_N$. Subsequent works have studied it over both $\mathbb{QR}_N$ [25] and $\mathbb{J}_N$ (the subgroup of elements of $\mathbb{Z}_N^*$ with Jacobi symbol +1) [20].

inputs. The SAGM imposes the stronger constraint that $\mathcal{A}$ output the *entire path of computation* (i.e., all intermediate group operations) resulting in output x. We show that if $\mathbb{QR}_N$ is modeled as a strongly algebraic group, then computing $x^{2^T} \bmod N$ using fewer than T squarings is as hard as factoring N. Our technique deviates substantially from standard proofs in the AGM, which work with groups of (known) prime order. Moreover, to the best of our knowledge, our result is the first formal argument supporting the sequential hardness of squaring in $\mathbb{QR}_N$. Our result immediately implies the security of Pietrzak's VDF [25] in the SAGM (assuming the hardness of factoring). We also give a negative result: we show that in the standard AGM, it is not possible to reduce hardness of speeding up sequential squaring to factoring, if factoring is hard.

Non-malleable timed commitments. The second part of our paper is concerned with strengthening the definition and security of *non-interactive timed commitments* (NITICs). A timed commitment differs from a regular one in that it additionally has a "forced" decommit routine that can be used to force open the commitment in case the committer refuses to open it. Moreover, a commitment comes with a proof that it can be forced open if needed. In Section 7 we introduce a strong notion of non-malleability for such schemes. To construct a non-malleable NITIC, we formalize as a stepping stone a timed public-key analogue that we call *time-released public-key encryption* (TRPKE). We show how to achieve an appropriate notion of CCA-security for TRPKE in both the ROM and the AGM model. We then show a generic transformation from CCA-secure TRPKE to non-malleable NITIC. Although our main purpose for introducing TRPKE is to obtain a non-malleable NITIC, we believe that TRPKE is an independently interesting primitive worthy of further study.

1.1 Related Work

We have already mentioned most of the related work about time-locked puzzles and similar primitives. Bitansky et al. [3] show an alternative approach to building time-locked puzzles from randomized encodings [2]. Mahmoody et al. [19] show constructions of time-locked puzzles in the random-oracle model. In very recent work, Malavolta and Thyagarajan [20] study a homomorphic variant of time-locked puzzles. Another line of work initiated by May [22] and later formalized by Rivest et al. [26] studies a model for timed message transmission between a sender and a receiver in the presence of a trusted server. Bellare and Goldwasser [1] studied a notion of "partial key escrow" in which a server can store keys in escrow, but only can learn some of them without expending significant computational effort. They also propose a time-locked puzzle from heuristic assumptions. Later works in this direction [9, 10] refine both the approaches and the models suggested in those works. Liu et al. [18] propose a time-released encryption scheme based on witness encryption in a model with a global trusted clock.

1.2 Overview of the Paper

We introduce notation and basic definitions in Section 2. In Section 3 we introduce the SAGM and state our hardness result about the sequential squaring assumption. We define TRPKE in Section 5 and give a construction in the AGM in Section 6. In Section 7, we first formalize NITIC and then state our generic instantiation for a non-malleable NITIC from TRPKE. For space restrictions, some of the technical parts of our paper are deferred to the appendix.

2 Notation and Preliminaries

Notation. Throughout the rest of the paper, we use ":=" to denote a deterministic assignment, and "$\leftarrow$" to denote a randomized assignment. In particular, "$x \leftarrow S$" denotes sampling a uniform element x from a set S. We denote the length of a bitstring x by $|x|$, and the length of the binary representation of an integer n by $||n||$. For a randomized algorithm $\mathcal{A}$, we let $\{\mathcal{A}(x)\}$ be the set of all possible outputs of $\mathcal{A}$ on input x. We denote the security parameter by κ. We write $\mathcal{G}^{\mathcal{A}}$ for the output of game $\mathcal{G}$ in which $\mathcal{A}$ is the adversary.

Running time. We consider running times of algorithms in some unspecified (but fixed) computational model, e.g., the Turing machine model. This is done both for simplicity of exposition and generality of our results. To simplify things further, we omit from our running-time analyses additive terms resulting from bitstring operations or passing arguments between algorithms, and we scale units so that multiplication in $\mathbb{QR}_N$ takes unit time. All algorithms are assumed to have arbitrary parallel computing resources.

The quadratic residue group $\mathbb{QR}_N$. Let GenMod be an algorithm that, on input 1^κ, outputs (N, p, q) where $N = pq$ and $p \neq q$ are two safe primes (i.e., $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are also prime) with $||p|| = ||q|| = \tau(\kappa)$, where $\tau(\kappa)$ is defined such that the fastest factoring algorithm takes time $\Theta(2^\kappa)$ to factor N with constant probability. GenMod may fail with negligible probability, but we ignore this from now on. It is well known that $\mathbb{QR}_N$ is cyclic with $|\mathbb{QR}_N| = \frac{\phi(N)}{4} = \frac{(p-1)(q-1)}{4}$.

Definition 1 (Factoring Problem). *For our purposes, we define the* factoring problem relative to GenMod *via the following game* $\mathbf{FAC}^{\mathcal{A}}_{\mathsf{GenMod}}$*:*

- ***Setup:*** $\mathbf{FAC}^{\mathcal{A}}_{\mathsf{GenMod}}$ *samples parameters* $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$*. It then runs* $\mathcal{A}$ *on input* N*.*
- ***Output:*** $\mathcal{A}$ *outputs positive integers* $p, q \notin \{1, N\}$*. Then* $\mathbf{FAC}^{\mathcal{A}}_{\mathsf{GenMod}}$ *outputs* 1 *if* $N = pq$*, and* 0 *otherwise.*

We say that the factoring problem is (ϵ, t)-hard relative to GenMod *if for all adversaries* $\mathcal{A}$ *running in time* t*,*

$$\Pr\left[\mathbf{FAC}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] \leq \epsilon.$$

The repeated squaring algorithm. In this work, we extensively make use of the idea of repeated squaring and multiplying; this offers a simple, efficient way to compute g^x given g in $\lceil \log x \rceil$ steps.[2] As a particularly important example, to compute g^{2^T} in T steps, the algorithm computes, in sequence, the values $g^2, g^4, \ldots, g^{2^{T-1}}, g^{2^T}$ (all modulo N, for $N \leftarrow \mathsf{GenMod}(1^\kappa)$), i.e., without any "multiplying" steps at all. We denote by RepSqr the algorithm that computes and outputs g^x on input (g, N, x) in this manner.

Sampling a random element in $\mathbb{QR}_N$ can be done by sampling $x \leftarrow \{0, \ldots, |\mathbb{QR}_N| - 1\}$ and running $\mathsf{RepSqr}(g, N, x)$ (this assumes that $|\mathbb{QR}_N|$ and hence factorization of N are known). If $|\mathbb{QR}_N|$ is unknown, we can sample $x \leftarrow \mathbb{Z}_{N^2}$ instead, which results in a negligible statistical difference in the output of $\mathsf{RepSqr}(g, N, x)$, which runs in at most

$$\lceil \log x \rceil \leq \lceil \log N^2 \rceil \leq 4\tau(\kappa)$$

[2] For positive i, a parallel algorithm can compute $g^{2^{i-1}+1}, \ldots, g^{2^i}$ simultaneously in step i, hence computing g^x (for any positive x) within $\lceil \log x \rceil$ steps.

steps. We omit this error for simplicity from here on, and denote by $\theta(\kappa) = 4\tau(\kappa)$ the upper bound of the running time of $\mathsf{RepSqr}(g, N, x)$ when $x \in \mathbb{Z}_{N^2}$. Additional primitives we use in this paper are defined in Appendix A.

2.1 The RSW Problem (with Preprocessing)

Definition 2 (RSW Problem). *The* T-RSW problem (with prepocessing) relative to GenMod *is defined via the following game* $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$*:*

- ***Setup:*** $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ *samples parameters* $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$.
- ***Preprocessing Phase:*** $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ *runs* $\mathcal{A}$ *on input* N.
- ***Online Phase:*** *When* $\mathcal{A}$ *outputs "online",* $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ *samples* $g \leftarrow \mathbb{QR}_N$ *and returns* g.
- ***Output Determination:*** *When* $\mathcal{A}$ *outputs* $\mathbf{X} \in \mathbb{QR}_N$, $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ *outputs* 1 *if* $\mathbf{X} = g^{2^T}$, *and* 0 *otherwise.*

The T*-RSW problem is* (ϵ, t_p, t_o)-hard relative to GenMod *if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase,*

$$\Pr\left[\mathbf{T\text{-}RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] \leq \epsilon.$$

Clearly, an adversary can run $\mathsf{RepSqr}(g, N, 2^T)$ to win the $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ game with probability 1. This means there is a threshold t^* (which is roughly the time for T group operations) such that the T-RSW problem is easy when $t_o \geq t^*$ (i.e., there is an adversary (ϵ, t_p, t_o)-breaking the T-RSW problem for any $\epsilon < 1$). In Section 3.1 we show that in the strong algebraic group model, the T-RSW problem is (ϵ, t_p, t_o)-hard (for negligible ϵ) when $t_o < t^*$ and $t_p = o(2^\kappa)$ is less than the time required to factor N. To put it another way, the fastest way to compute g^{2^T} (short of factoring N) is to run $\mathsf{RepSqr}(g, N, 2^T)$.

It is easy to see that hardness of RSW implies hardness of factoring. Concretely, if there is an adversary $\mathcal{A}$ (ϵ, t)-breaking the factoring problem, then a T-RSW (for $T \gg \kappa$ and $T \gg t$) adversary can run $(p, q) \leftarrow \mathcal{A}(N)$, compute $\phi := \phi(N) = (p-1)(q-1)$ and $z := [2^T \mod \frac{\phi}{4}]$, and output g^z, winning $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ with probability ϵ in time $t + \theta(\kappa) \ll T$.

3 Algebraic Hardness of the RSW Problem

In this section we explain how, in the AGM, the RSW assumption can be related to the factoring assumption. To this end, we first briefly recall the AGM, and then introduce a refinement of the AGM, called the *strong algebraic group model* (SAGM), which lies in between the GGM and the AGM. In the subsequent section, we show that one cannot reduce the hardness of RSW from the hardness of factoring in the plain AGM with respect to algebraic reductions.

3.1 The Strong Algebraic Group Model

The *algebraic group model* (AGM), introduced by Fuchsbauer, Kiltz, and Loss [12], lies between the generic group model (GGM) and the standard model. As in the standard model, the adversary is given actual group elements, rather than their encodings as in the GGM. Hence, the AGM covers adversaries that exploit the *representations* of group elements (via, e.g., computing the Jacobi symbol), a much wider category than those covered by the GGM. We first recall the notion of an

algebraic algorithm [7,24]. In the following, assume that $\mathbb{G}$ is the description of a cyclic group that has been output previously by a suitable group generation procedure. For our purposes, $\mathbb{G}$ will always be $\mathbb{QR}_N$, where N is generated via $N \leftarrow \mathsf{GenMod}(1^\kappa)$.

Definition 3 (Algebraic Algorithm). *An algorithm $\mathcal{A}$ over $\mathbb{G}$ is called* algebraic *if, whenever $\mathcal{A}$ outputs $\mathbf{X} \in \mathbb{G}$,[3] it also outputs a vector $\boldsymbol{\lambda}$ s.t. $\mathbf{X} = \prod_i L_i^{\lambda_i}$, where $\boldsymbol{L}$ denotes the list of group elements that $\mathcal{A}$ has received as inputs up to that point.*

In the AGM, all algorithms are treated as algebraic. In the original formulation of the AGM, the authors considered prime-order groups, whereas we consider the composite order (albeit cyclic) group $\mathbb{QR}_N$. Another distinction from their original work is that we consider a setting in which the group may vary, whereas they consider a setting with a fixed group. The AGM does not support the concept of "number of steps (i.e., group operations)" or "time" of computing a group element. As we will see, if we only consider whether an adversary is polynomial-time or not, this does not cause a problem, as all group elements output by an algebraic adversary can be computed in polynomial steps. However, for time-sensitive assumptions such as hardness of the RSW problem (recall that RSW becomes easy for an adversary making T group operations online), it is not sufficient for use: given challenge g, an algebraic adversary can simply output g^{2^T} together with $\boldsymbol{\lambda} = (2^T)$, and there is no way to analyze its running time in the AGM (other than counting the number of output group elements, which is 1 in this case). (This point is made more formal in Appendix 4.) We develop a stronger model which we call the *strong AGM (SAGM)*, which captures the running time of an algebraic algorithm via counting its group operations.

The Strong AGM. Recall that in the AGM, whenever an algorithm outputs a group element $\mathbf{X}$, it is required to also provide an algebraic representation of $\mathbf{X}$, using all group elements it has received so far. In the SAGM we require an algorithm to provide a *multiplicative* representation of the output group element, using all group elements it has received so far and its previous outputs.

Definition 4 (Strong AGM). *Algorithm $\mathcal{A}$ over $\mathbb{G}$ is called* strongly algebraic *if in each step $\mathcal{A}$ outputs one or more tuples $(\mathbf{X}, \mathbf{X}_1, \mathbf{X}_2) \in \mathbb{G}^3$, where $\mathbf{X} = \mathbf{X}_1\mathbf{X}_2$ and $\mathbf{X}_1, \mathbf{X}_2$ were either previously received by $\mathcal{A}$ or output by $\mathcal{A}$ in previous steps. In the strong AGM, all algorithms are treated as strongly algebraic.*

We allow an algorithm in the SAGM to output multiple new group elements in a step, as long as all of them can be written in terms of *previously* received or output group elements. This captures what a parallel algorithm can do.[4] Summarizing, there are two fundamental differences between the AGM and the SAGM: first, an algorithm in the SAGM has to output all its intermediate steps for computing a group element; second, the AGM allows multiple sequential multiplications/exponentiations to be done in one step, whereas the SAGM only allows a single multiplication to be done (although many such multiplications can be done in parallel). As an example of a strongly algebraic algorithm, given n group elements $\mathbf{X}_1, \ldots, \mathbf{X}_n$, the following algorithm $\widetilde{\mathsf{Mult}}$ outputs $\mathbf{X}_1 \cdots \mathbf{X}_n$ in $\lceil \log n \rceil$ steps: If $n = 1$ then $\widetilde{\mathsf{Mult}}(\mathbf{X}_1)$ outputs $\mathbf{X}_1$, otherwise $\widetilde{\mathsf{Mult}}(\mathbf{X}_1, \ldots, \mathbf{X}_n)$ runs $\mathbf{Y} := \widetilde{\mathsf{Mult}}(\mathbf{X}_1, \ldots, \mathbf{X}_{\lceil n/2 \rceil})$ and $\mathbf{Z} := \widetilde{\mathsf{Mult}}(\mathbf{X}_{\lceil n/2 \rceil + 1}, \ldots, \mathbf{X}_n)$ in parallel, and outputs $(\mathbf{YZ}, \mathbf{Y}, \mathbf{Z})$. (Tilde denotes that the algorithm is strongly algebraic.) As another example, in repeated squaring every step computes a new group element as the product of two

[3] Following [12], we write elements of $\mathbb{G}$ (except for the fixed generator g) in bold, upper-case letters.

[4] If we instead required the algorithm to output a single element in each step, this would rule out parallelism.

known group elements, so it can be naturally turned into an algorithm in the SAGM. We use $\widetilde{\mathsf{RepSqr}}$ to denote such "strongly algebraic version" of repeated squaring. As mentioned in Section 2, $\widetilde{\mathsf{RepSqr}}(g, x)$ computes g^x in $\lceil \log x \rceil$ steps.

Comparison with the GGM and AGM. In the same way that the AGM lies between the GGM and the standard model, the SAGM lies in between the GGM and the AGM. It is easy to construct a strongly algebraic algorithm that operates in a group specific way (say, by computing a Jacobi symbol over the quadratic residue group) and hence is not generic. The following theorem shows that any algebraic algorithm (whose running time is measured via number of output group elements) can be turned into a strongly algebraic algorithm with a polylogarithmic time loss (assuming that the output length is polynomial). Therefore, if we only consider the asymptotic behavior of polynomial-time algorithms, then the SAGM and the AGM are equivalent.

Theorem 1. *Suppose $\mathcal{A}$ is an algebraic algorithm over $\mathbb{G}$, which outputs $\mathbf{X} \in \mathbb{G}$ together with its algebraic representation $(\lambda_1, \ldots, \lambda_n)$. (That is, $\mathbf{X} = \mathbf{X}_1^{\lambda_1} \cdots \mathbf{X}_n^{\lambda_n}$ where $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \mathbb{G}$ are $\mathcal{A}$'s inputs.) Then there exists a strongly algebraic algorithm $\mathcal{B}^{\mathcal{A}}$ in $\mathbb{G}$ (whose inputs are the same as $\mathcal{A}$'s) which also outputs $\mathbf{X}$ in $\max\{\lceil \log \lambda_1 \rceil, \ldots, \lceil \log \lambda_n \rceil\} + \lceil \log n \rceil$ steps.*

Proof. Consider the following strongly algebraic algorithm $\mathcal{B}(\mathbf{X}_1, \ldots, \mathbf{X}_n)$:

1. Run $\mathcal{A}(\mathbf{X}_1, \ldots, \mathbf{X}_n)$ and receive $\mathcal{A}$'s output $\mathbf{X}$ together with $(\lambda_1, \ldots, \lambda_n)$.
2. Run $\mathbf{X}_1^{\lambda_1} := \widetilde{\mathsf{RepSqr}}(\mathbf{X}_1, \lambda_1), \ldots, \mathbf{X}_n^{\lambda_n} := \widetilde{\mathsf{RepSqr}}(\mathbf{X}_n, \lambda_n)$ in parallel.
3. Run $\widetilde{\mathsf{Mult}}(\mathbf{X}_1^{\lambda_1}, \ldots, \mathbf{X}_n^{\lambda_n})$.

Clearly, $\mathcal{B}$'s last output contains $\mathbf{X}_1^{\lambda_1} \cdots \mathbf{X}_n^{\lambda_n} = \mathbf{X}$. Running $\widetilde{\mathsf{RepSqr}}(\mathbf{X}_1, \lambda_1), \ldots, \widetilde{\mathsf{RepSqr}}(\mathbf{X}_n, \lambda_n)$ in parallel costs $\mathcal{B}$ $\max\{\lceil \log \lambda_1 \rceil, \ldots, \lceil \log \lambda_n \rceil\}$ steps; running $\widetilde{\mathsf{Mult}}(\mathbf{X}_1^{\lambda_1}, \ldots, \mathbf{X}_n^{\lambda_n})$ costs $\mathcal{B}$ $\lceil \log n \rceil$ steps. So $\mathcal{B}$ runs in $\max\{\lceil \log \lambda_1 \rceil, \ldots, \lceil \log \lambda_n \rceil\} + \lceil \log n \rceil$ steps in total.

Running time in the SAGM. Recall that hardness of the T-RSW problem is parameterized by the advantage ϵ and the running time t. In the SAGM running time is counted in two ways: the number of "steps" (i.e., group operations), and the "normal" running time as measured in some underlying computational model (e.g., the Turing machine model). The exact relation between them is determined by the computational model. We define the running time as a *pair*, so "running time is (t_1, t_2)" is interpreted as carrying out t_1 group operations and running in time t_2.

3.2 Hardness of the RSW Problem in the Strong AGM

We prove hardness of the RSW problem in the SAGM. Theorem 2 states that solving the T-RSW problem in fewer than T steps is as hard as factoring.

Factoring with a known multiple of $\phi(N)$. We will use the following well-known result which states that N can be efficiently factored given a positive multiple of $\phi(N)$. The lemma follows from a straightforward adaption of the proof for Theorem 8.50 in [14]. We will use the following folklore factoring procedure that can be applied whenever $N = p \cdot q$ and a multiple $m > 0$ of $\phi(N)$ are known.

Lemma 1. *Suppose $N \leftarrow \mathsf{GenMod}(1^\kappa)$ and $m = \alpha \cdot \phi(N)$ (where $\alpha \in \mathbb{Z}^+$). Then there exists an algorithm $\mathsf{Factor}(N, m)$ which runs in time at most $4\lceil \log \alpha \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil$ and outputs $p', q' \notin \{1, N\}$ s.t. $N = p'q'$ with probability at least $\frac{1}{2}$.*

Theorem 2. *Assume that factoring is* $(\epsilon, t_p + t_o + \theta(\kappa) + 4\lceil \log \alpha \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil)$*-hard relative to* GenMod*. Let* T *be any positive integer. Then the* T*-RSW problem is* $(2\epsilon, (0, t_p), (T-1, t_o))$*-hard relative to* GenMod *in the SAGM.* [5]

Proof. Let $\mathcal{A}$ be a strongly algebraic algorithm that runs in time $(0, t_p)$ in the preprocessing phase and $(T-1, t_o)$ in the online phase. That is, in the preprocessing phase, $\mathcal{A}$ runs in time t_p but does not output any group element (note that in the preprocessing phase $\mathcal{A}$ is not given any group element, so by definition of the SAGM, $\mathcal{A}$ cannot output any group element); in the online phase $\mathcal{A}$ runs in time t_o and can do at most $T-1$ group operations. For $\mathcal{A}$'s challenge g (given at the beginning of the online phase of $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$), and any $\mathbf{X} \in \mathbb{QR}_N$ output by $\mathcal{A}$ during the online phase of $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$, we recursively define the *discrete logarithm* of $\mathbf{X}$ (with respect to $\mathcal{A}$ and g), $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) \in \mathbb{Z}^+$, as follows:

- $\mathsf{DL}_{\mathcal{A}}(g, g) = 1$;
- If $\mathcal{A}$ outputs $(\mathbf{X}, \mathbf{X}_1, \mathbf{X}_2)$ in a certain step, then $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) = \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1) + \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2)$.

Obviously, $g^{\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})} = \mathbf{X}$ for any $\mathbf{X} \in \mathbb{QR}_N$ output by $\mathcal{A}$.

Lemma 2. *Let* s *be a positive integer. For any strongly algebraic algorithm* $\mathcal{A}$ *running in at most* s *steps, all of* $\mathcal{A}$*'s outputs have discrete logarithm at most* 2^s*.*

Proof. The proof is by induction on s. If $s = 1$, then the only group elements $\mathcal{A}$ can output are g and g^2, so the lemma holds. Now suppose that the lemma holds for $s-1$. This means that all of $\mathcal{A}$'s outputs in steps $1, \ldots, s-1$ have discrete logarithm at most 2^{s-1}. Suppose $\mathcal{A}$ outputs $(\mathbf{X}, \mathbf{X}_1, \mathbf{X}_2)$ in step s. Then $\mathbf{X}_1$ and $\mathbf{X}_2$ must be g or one of $\mathcal{A}$'s outputs in steps $1, \ldots, s-1$. We have that $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) = \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1) + \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2) \leq 2^{s-1} + 2^{s-1} = 2^s$, so the lemma holds for s as well.

Going back to the theorem, we construct a reduction $\mathcal{R}$ which factors N as follows. $\mathcal{R}$, on input N, runs $\mathcal{A}(N)$. When $\mathcal{A}$ outputs "online", $\mathcal{R}$ samples $g \leftarrow \mathbb{QR}_N$ and sends g to $\mathcal{A}$. If $\mathcal{A}$ outputs $\mathbf{X}$ together with a flag (indicating that $\mathbf{X}$ is supposed to be g^{2^T}), $\mathcal{R}$ (recursively) computes $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})$, and outputs $\mathsf{Factor}(N, 4(2^T - \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})))$.

By Lemma 2, $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) < 2^T$, so $4(2^T - \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})) > 0$. Furthermore, if $\mathcal{A}$ wins, i.e., $\mathbf{X} = g^{2^T}$, we have $g^{2^T - \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})} = 1$, so $|\mathbb{QR}_N| = \frac{\phi(N)}{4} \mid \left(2^T - \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})\right)$ (here we need the fact that $\mathbb{QR}_N$ is cyclic), so $4(2^T - \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}))$ is a multiple of $\phi(N)$. By Lemma 1, $\mathcal{R}$ runs in time at most $t_p + t_o + \theta(\kappa) + 4\lceil \log \alpha \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil$, and successfully factors N with probability at least $\frac{1}{2}$. We conclude that $\Pr\left[\mathbf{T\text{-}RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] \leq 2\epsilon$, which completes the proof.

4 The RSW Problem in the AGM

We show that the AGM is insufficient to reduce the factoring assumption to the T-RSW assumption (assuming factoring is hard to begin with). This, combined with Theorem 2, forms a separation result between the AGM and the SAGM. On the other hand, as we have seen in Theorem 1, the AGM and the SAGM are equivalent if we only care about the asymptotic behavior of the adversary. Specifically, we give a metareduction $\mathcal{M}$ that converts any efficient algebraic reduction $\mathcal{R}$ into an

[5] To be precise, since the strongly algebraic $\mathcal{A}$ needs to output a number of group elements, we require $\mathcal{A}$ to output a flag together with its final output, which is supposed to be g^{2^T}. That is, when $\mathcal{A}$ outputs $\mathbf{X} \in \mathbb{QR}_N$ together with a flag, $\mathcal{A}$ wins the T-RSW game $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$ if $\mathbf{X} = g^{2^T}$.

efficient algorithm for factoring. Below, we use $\mathcal{R}^{\mathcal{A}}$ to denote that $\mathcal{R}$ can run $\mathcal{A}$ as many times in the AGM on any choice of $\mathcal{R}$'s random coins. It must provide all of $\mathcal{A}$'s random coins (as well as $\mathcal{A}$'s input) upon invoking $\mathcal{A}$. In particular, $\mathcal{R}$ may not rewind (or restart) $\mathcal{A}$ to intermediary points of its execution or write to $\mathcal{A}$'s input tape while it is running $\mathcal{A}$.

Theorem 3. *Let $\mathcal{R}$ be a (reduction) algorithm s.t. for any algorithm $\mathcal{A}$ that runs in time t_p in the preprocessing phase and t_o in the online phase of game $\mathbf{T\text{-}RSW}_{\mathsf{GenMod}}$, and for which $\Pr\left[\mathbf{T\text{-}RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] \leq \epsilon$, the algorithm $\mathcal{B} := \mathcal{R}^{\mathcal{A}}$ runs in time at most t' and satisfies $\Pr\left[\mathbf{FAC}^{\mathcal{B}}_{\mathsf{GenMod}} = 1\right] > \epsilon'$. Then there exists an algorithm $\mathcal{M}$ that runs in time at most $(Q+1)\cdot T + t'$ and satisfies $\Pr\left[\mathbf{FAC}^{\mathcal{M}}_{\mathsf{GenMod}} = 1\right] > \epsilon'$, where $\mathcal{R}$ runs $\mathcal{A}$ Q times.*

Proof. Let $\mathcal{R}$ be as described in the theorem statement. Consider the following (metareduction) algorithm $\mathcal{M}$ that plays in game $\mathbf{FAC}_{\mathsf{GenMod}}$:

- $\mathcal{M}$, on input N, chooses some (fixed) random coins ρ for $\mathcal{R}$.
- For $i = 1, \ldots, Q$,
  - $\mathcal{M}$ runs $\mathcal{R}(N; \rho)$.
  - For $j = 1, \ldots, i-1$, $\mathcal{M}$ simulates the behavior of $\mathcal{A}$ in $\mathcal{R}$'s j-th run as follows: when $\mathcal{R}$ outputs N_j, $\mathcal{M}$ replies with "online" immediately; when $\mathcal{R}$ outputs g_j, $\mathcal{M}$ replies with $\mathbf{X}_j$ immediately. (If $i = 1$ then skip this step.)
  - In $\mathcal{R}$'s i-th run of $\mathcal{A}$, $\mathcal{M}$ computes $\mathbf{X}_i := \mathsf{RepSqr}(g_i, N_i, 2^T)$ and restarts $\mathcal{R}(N; \rho)$.
- When $\mathcal{R}$ outputs factors p, q of N, $\mathcal{M}$ forwards them as the result to its own challenger in $\mathbf{FAC}_{\mathsf{GenMod}}$.

Note that the simulated adversary ignores the random coins that $\mathcal{R}$ might provide to it, as its behavior is deterministic. The simulated adversary always wins the game $\mathbf{FAC}_{\mathsf{GenMod}}$, since in its j-th run, its challenge is (N_j, g_j) and it outputs $\mathbf{X}_j = \mathsf{RepSqr}(g_j, N_j, 2^T)$. By assumption, $\mathcal{B} = \mathcal{R}^{\mathcal{A}}$ runs in time at most t' and satisfies $\Pr\left[\mathbf{FAC}^{\mathcal{B}}_{\mathsf{GenMod}} = 1\right] > \epsilon'$, given that $\mathcal{A}$ runs in time at most t and satisfies $\Pr\left[\mathbf{T\text{-}RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] > \epsilon$ (which is the case above since the $\mathcal{A}$ simulated by $\mathcal{M}$ runs in time 0 and wins with probability 1). $\mathcal{M}$ restarts $\mathcal{R}$ Q times (i.e., runs $\mathcal{R}$ $Q+1$ times). Thus, $\mathcal{M}$ runs in time at most $(Q+1)\cdot T + t'$ and $\Pr\left[\mathbf{FAC}^{\mathcal{M}}_{\mathsf{GenMod}} = 1\right] > \epsilon'$.

5 Definitions for Time-Released Public-Key Encryption

In this section, we define the syntax and security of a *time-released public-key encryption (TRPKE)* scheme, which will serve as a central ingredient to our construction of a non-malleable timed commitment scheme in the subsequent sections. TRPKE can be thought of the time-released analogue to the standard notion of public-key encryption. The main difference is that a ciphertext can now forced open within some (predefined) time by anybody. However, the owner of the secret key should still be able to decrypt much faster.

Definition 5 (Time-Released Public-Key Encryption Scheme). *A (t_e, t_{fd}, t_{sd})-*time-released public-key encryption (TRPKE) scheme *is a tuple of algorithms $\mathsf{TRPKE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ with the following behavior:*

- *The randomized* parameter generation algorithm PGen *takes as input the security parameter 1^κ and outputs public parameters par.*

- *The randomized* key generation algorithm KGen *takes as input public parameters par and outputs a pair of keys* (sk, pk)*. We assume, for simplicity, that for* pk, sk *generated as* $(sk, pk) \leftarrow \mathsf{KGen}(par)$*, the public key pk contains par and the secret key sk contains the public key pk.*
- *The randomized* encryption algorithm Enc *takes as input a public key pk and a message* $m \in \{0,1\}^*$*. It outputs a ciphertext* $c \in \{0,1\}^*$ *and runs in time at most* t_e *for all* m*.*
- *The deterministic* fast decryption algorithm Dec_f *takes as input a secret key sk and a ciphertext* $c \in \{0,1\}^*$*. It outputs a message* $m \in \{0,1\}^*$ *or* $\perp$ *and runs in time at most* t_{fd} *for all c.*
- *The deterministic* slow decryption algorithm Dec_s *takes as input a public key pk and a ciphertext* $c \in \{0,1\}^*$*. It outputs a message* $m \in \{0,1\}^*$ *or* $\perp$ *and runs in time at least* t_{sd} *for all c.*

As we will see, we are usually interested in schemes for which $t_{fd} < t_{sd}$, i.e., the running time of fast decryption is always faster than the running time of slow decryption. Correctness requires that decrypting a ciphertext (no matter in the "fast" or "slow" way) always yields the original message that was encrypted.

Definition 6 (Perfect Correctness of TRPKE). *A TRPKE scheme* $\mathsf{TRPKE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *satisfies* perfect correctness *if for all* $m \in \{0,1\}^*$*, all* $par \in \{\mathsf{PGen}(1^\kappa)\}$*, all* $(sk, pk) \in \{\mathsf{KGen}(par)\}$*, and all* $c \in \{\mathsf{Enc}(pk, m)\}$*, it holds that* $m = \mathsf{Dec}_f(sk, c) = \mathsf{Dec}_s(pk, c)$*.*

We next introduce a strong security notion for TRPKE that resembles the standard security under Chosen Ciphertext Attacks (CCA) notion for public-key and secret-key encryption. (Defining an analogous CPA security notion would be straightforward, but we do not consider this definition in our work.) As in standard public-key encryption, the adversary $\mathcal{A}$ in security game $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$ is asked to provide messages m_0, m_1 and gets a *challenge ciphertext* computed via Enc on either one of them, with probability $\frac{1}{2}$. $\mathcal{A}$ gets access to a *decryption oracle* DEC that allows the adversary to decrypt arbitrary ciphertexts of its choice (except for the challenge ciphertext). To make DEC "useful" to $\mathcal{A}$, we require that it decrypts using the fast decryption algorithm Dec_f — if it used, instead, Dec_s, the adversary could simply decrypt itself.

$\underline{\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TRPKE}}}$:

- **Setup:** $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$ samples parameters par via $par \leftarrow \mathsf{PGen}(1^\kappa)$ and a bit $b \leftarrow \{0,1\}$. It then samples a pair of keys (sk, pk) as $(sk, pk) \leftarrow \mathsf{KGen}(par)$.
- **Preprocessing Phase:** $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$ runs $\mathcal{A}$ on input pk. In this phase, $\mathcal{A}$ is given access to a *decryption oracle* DEC, which on input c, returns a plaintext $m := \mathsf{Dec}_f(sk, c)$ *in time* t_{fd}.
- **Challenge Query:** When $\mathcal{A}$ outputs (m_0, m_1), $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$ returns the challenge ciphertext $c_b \leftarrow \mathsf{Enc}(pk, m_b)$.
- **Online Phase:** After receiving c_b, $\mathcal{A}$ continues to have access to DEC, except that $\mathcal{A}$ may not query DEC on c_b.
- **Output Determination:** When $\mathcal{A}$ outputs a bit b', $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$ outputs 1 if $b' = b$, and 0 otherwise.

We define the CCA-security of a TRPKE scheme in the following way. To capture the fact that the challenge ciphertext will be trivially decryptable after time t_{sd}, we only consider those adversaries whose running time is less than t_{sd} in the online phase.

Definition 7 (CCA Security for TRPKE). *A TRPKE scheme* TRPKE *is* (ϵ, t_p, t_o)-CCA-secure *(where* $t_o < t_{sd}$*) if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase of* $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$*,*

$$\Pr\left[\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TRPKE}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

6 TRPKE Constructions in the AGM

In this section, we present constructions for TRPKE schemes in the algebraic group model. We begin by introducing the following problem:

Definition 8 (Decisional T-RSW Problem). *The* decisional T-RSW problem (with prepocessing) relative to GenMod *is defined via the following game* $\mathbf{T\text{-}DRSW}_{\mathsf{GenMod}}$*:*

- ***Setup:*** $\mathbf{T\text{-}DRSW}_{\mathsf{GenMod}}$ *samples parameters* $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$ *and a bit* $b \leftarrow \{0, 1\}$*.*
- ***Preprocessing Phase:*** $\mathbf{T\text{-}DRSW}_{\mathsf{GenMod}}$ *runs* $\mathcal{A}$ *on input* N*.*
- ***Online Phase:*** *When* $\mathcal{A}$ *outputs "online",* $\mathbf{T\text{-}DRSW}_{\mathsf{GenMod}}$ *samples* $g, \mathbf{X} \leftarrow \mathbb{QR}_N$*. If* $b = 0$*, it returns* $(g, \mathbf{X})$*, otherwise* (g, g^{2^T})*.*
- ***Output Determination:*** *When* $\mathcal{A}$ *outputs* b'*,* $\mathbf{T\text{-}DRSW}_{\mathsf{GenMod}}$ *outputs* 1 *if* $b' = b$*, and* 0 *otherwise.*

We say that the decisional T*-RSW problem is* (ϵ, t_p, t_o)-hard relative to GenMod *if for any adversary* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase,*

$$\Pr\left[\mathbf{T\text{-}DRSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

Our problem is related to the Generalized BBS (GBBS) assumption that was introduced by Boneh and Naor [6]. However, there are some key differences between the decisional T-RSW problem and the GBBS assumption: first, the adversary in the GBBS assumption gets to see the group elements $g, g^2, g^4, g^{16}, g^{256}, ..., g^{2^{2^k}}$ and is asked to distinguish $g^{2^{2^{k+1}}}$ from a random quadratic residue $\mathbf{X}$. Second, the adversary in the GBBS assumption does not get any preprocessing time. If we set $T = 2^{k+1}$ and allow for equal preprocessing time in both problem, then the hardness of the decisional T-RSW problem is implied by the GBBS assumption. On the other hand, it is unclear how to compare the two if preprocessing is allowed in the decisional T-RSW problem, but not in the GBBS assumption. It is also very similar to the strong sequential squaring assumption defined in [20]; however, in their version of the assumption the adversary is given a generator of the group in the preprocessing phase.

6.1 CCA-Secure TRPKE in the AGM

We present our construction of a CCA-secure TRPKE scheme in Fig. 1. Our construction follows the Naor-Yung paradigm [23].

Theorem 4. *Suppose that*

- NIZK *is* $(\epsilon_{ZK}, t_p + t_o + T + \theta(\kappa))$*-zero-knowledge and* $(\epsilon_{SS}, t_p + t_o + T + \theta(\kappa))$*-simulation sound, and*

For any $N = pq$ where $p \neq q$ are two safe primes for $||p|| = ||q||$, let $\mathsf{NIZK}_N = (\mathsf{GenZK}_N, \mathsf{Prove}_N, \mathsf{Vrfy}_N, \mathsf{SimGen}_N, \mathsf{SimProve}_N)$ be a $(t_{prove}, t_{vrfy}, t_{sgen}, t_{sprove})$-NIZK for relation

$$R_N = \{((\mathbf{R}_1, \mathbf{R}_2, \mathbf{X}_1, \mathbf{X}_2), \mathbf{W}) | \mathbf{X}_1 = \mathbf{R}_1^{2^T} \cdot \mathbf{W} \wedge \mathbf{X}_2 = \mathbf{R}_2^{2^T} \cdot \mathbf{W}\}$$

(where all operations are over $\mathbb{Z}_N$). Henceforth we drop the subscript N and write NIZK for NIZK_N, and so on.

- $\mathsf{Gen}(1^\kappa)$: Run $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$, compute $\phi := \phi(N) = (p-1)(q-1)$, and output $par := (N, \phi)$.
- $\mathsf{KGen}(N, \phi)$: Choose $g \leftarrow \mathbb{QR}_N$, run $crs \leftarrow \mathsf{GenZK}(1^\kappa)$ and output $(sk := (crs, N, \phi, g), pk := (crs, N, g))$.
- $\mathsf{Enc}((crs, N, g), \mathbf{M})$: Choose $r_1, r_2 \leftarrow \mathbb{Z}_{N^2}$ and compute, for $i = 1, 2$,

  $$\mathbf{R}_i := \mathsf{RepSqr}(g, N, r_i), \mathbf{Z}_i := \mathsf{RepSqr}(\mathbf{R}_i, N, 2^T), \mathbf{C}_i := \mathbf{Z}_i \cdot \mathbf{M},$$

  and $\pi \leftarrow \mathsf{Prove}(crs, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \mathbf{M})$. Output $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$.
- $\mathsf{Dec}_f((crs, N, \phi, g), (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: Compute $z := [2^T \mod \frac{\phi}{4}]$, $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{R}_1, N, z)$ and $\mathbf{M} := \frac{\mathbf{C}_1}{\mathbf{Z}_1}$. If $\mathsf{Vrfy}(crs, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 1$ then output $\mathbf{M}$, otherwise output $\perp$.
- $\mathsf{Dec}_s((crs, N, g), (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: Compute $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{R}_1, N, 2^T)$ and $\mathbf{M} := \frac{\mathbf{C}_1}{\mathbf{Z}_1}$. If $\mathsf{Vrfy}(crs, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 1$ then output $\mathbf{M}$, otherwise output $\perp$.

Fig. 1. A CCA-secure $(t_{prove} + 2\theta(\kappa), 2\theta(\kappa), T + t_{vrfy} + \theta(\kappa), T + t_{vrfy})$-TRPKE scheme

- *The decisional T-RSW problem is $(\epsilon_{DRSW}, t_p + T + t_{sgen} + \theta(\kappa), t_o + t_{sprove})$-hard relative to* GenMod.

Then the TRPKE scheme in Fig. 1 is $(\epsilon_{ZK} + \epsilon_{SS} + 2\epsilon_{DRSW}, t_p, t_o)$-CCA-secure in the AGM.

The proof of this theorem is very similar to that of CCA security of the Naor-Yung scheme. (Recall that the Naor-Yung scheme is CCA2-secure when using a simulation sound NIZK.) The main difference is that there must be a way to simulate the decryption oracle in a fast manner, which can be achieved in the AGM via preprocessing. Concretely, the simulator can compute $\mathbf{H} := g^{2^T}$ in the preprocessing phase, and when the adversary queries the decryption oracle on $(\mathbf{R}_1, \ldots)$, the simulator can compute $\mathbf{Z}_1 = \mathbf{R}_1^{2^T}$ as $\mathbf{H}^{r_1}$, where $\log_g \mathbf{R}_1$ can be computed using the algebraic representation that the adversary outputs. (All other steps in the decryption algorithm can be computed fast.)

Proof. Let $\mathcal{A}$ be an algebraic adversary in Game $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$, with preprocessing time t_p and online time t_o. Suppose $\mathcal{A}$'s challenge ciphertext is $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$. The proof goes by a sequence of games, which we describe next.

$\mathcal{G}_0$: $\mathcal{G}_0$ is the original CCA Security Game, $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$. We assume that the game challenger computes the discrete logarithm between g and all group elements which appear in the game; in particular, since $\mathcal{A}$ is algebraic, and all group elements $\mathcal{A}$ receives are g and DEC oracle outputs, the game challenger can eventually compute the discrete logarithm between g and all group elements $\mathcal{A}$ outputs according to the algebraic coefficients $\mathcal{A}$ outputs.

$\mathcal{G}_1$: $\mathcal{G}_1$ is identical to $\mathcal{G}_0$, except that $\mathbf{H} := \mathsf{RepSqr}(g, N, 2^T)$ is computed at the beginning of the game (so $\mathbf{H} = g^{2^T}$), and when $\mathcal{A}$ queries $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, let $r_1 := \log_g \mathbf{R}_1$ and $\mathbf{Z}_1 :=$

$\mathsf{RepSqr}(\mathbf{H}, N, r_1)$ (instead of $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{R}_1, N, 2^T)$). Since

$$\mathbf{Z}_1 = \mathbf{R}_1^{2^T} = g^{2^T \cdot r_1} = \mathbf{H}^{r_1},$$

this is merely a conceptual change, so

$$\Pr[\mathcal{G}_1^{\mathcal{A}} = 1] = \Pr[\mathcal{G}_0^{\mathcal{A}} = 1].$$

Note that $\mathcal{G}_1$ spends time T computing $\mathbf{H}$, and after that, a decryption oracle DEC query is answered in time roughly $\theta(\kappa)$. The purpose of $\mathcal{G}_1$ is to move the slow computation while answering DEC queries to preprocessing, so that DEC queries can be answered in a fast way.

$\mathcal{G}_2$: $\mathcal{G}_2$ is identical to $\mathcal{G}_1$, except that crs and π^* are simulated. That is, in Gen run $(crs, td) \leftarrow \mathsf{SimGen}(1^\kappa)$, and in the challenge ciphertext compute $\pi^* \leftarrow \mathsf{SimProve}((\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*), td)$.

We upper bound $|\Pr[\mathcal{G}_2^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_1^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{ZK}$ to the zero-knowledge property of NIZK. $\mathcal{R}_{ZK}$ runs the code of $\mathcal{G}_2$, except that it publishes the CRS from the zero-knowledge challenger, and uses the zero-knowledge proof from the zero-knowledge challenger as part of the challenge ciphertext. Concretely, $\mathcal{R}_{ZK}$ works as follows:

- Setup: $\mathcal{R}_{ZK}$, on input crs^*, chooses $g \leftarrow \mathbb{QR}_N$ and computes $\mathbf{H} := \mathsf{RepSqr}(g, N, 2^T)$; chooses $r_1^*, r_2^* \leftarrow \{0, \ldots, |\mathbb{QR}_N| - 1\}$ and computes

  $$\mathbf{R}_1^* := \mathsf{RepSqr}(g, N, r_1^*), \mathbf{R}_2^* := \mathsf{RepSqr}(g, N, r_2^*),$$

  $$\mathbf{Z}_1^* := \mathsf{RepSqr}(\mathbf{H}, N, r_1^*), \mathbf{Z}_2^* := \mathsf{RepSqr}(\mathbf{H}, N, r_2^*),$$

  (note that $\mathbf{R}_1^*$, $\mathbf{R}_2^*$, $\mathbf{Z}_1^*$, $\mathbf{Z}_2^*$ can be computed in parallel). Then $\mathcal{R}_{ZK}$ runs $\mathcal{A}(N, g, crs^*)$. $\mathcal{R}_{ZK}$ answers $\mathcal{A}$'s DEC queries as described in $\mathcal{G}_1$. That is, on $\mathcal{A}$'s query $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, $\mathcal{R}_{ZK}$ computes $r_1 = \log_g \mathbf{R}_1$ according to the algebraic coefficients $\mathcal{A}$ outputs, $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{H}, N, r_1)$, and $\mathbf{M} := \dfrac{\mathbf{C}_1}{\mathbf{Z}_1}$; if $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 1$ then $\mathcal{R}_{ZK}$ returns $\mathbf{M}$, otherwise $\mathcal{R}_{ZK}$ returns $\perp$.
- Online phase: When $\mathcal{A}$ makes its challenge query on $(\mathbf{M}_0, \mathbf{M}_1)$, $\mathcal{R}_{ZK}$ chooses $b \leftarrow \{0, 1\}$ and

  $$\mathbf{C}_1^* := \mathbf{Z}_1^* \cdot \mathbf{M}_b, \mathbf{C}_2^* := \mathbf{Z}_2^* \cdot \mathbf{M}_b,$$

  $$\pi^* \leftarrow \mathsf{PROVE}((\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*), \mathbf{M}_b),$$

  and outputs $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$. After that, $\mathcal{R}$ answers $\mathcal{A}$'s DEC queries just as in setup.
- Output: On $\mathcal{A}$'s output bit b', $\mathcal{R}_{ZK}$ outputs 1 if $b' = b$, and 0 otherwise.

Clearly, $\mathcal{R}_{ZK}$ runs in time $t_p + t_o + T + \theta(\kappa)$ (i.e., $t_p + T + \theta(\kappa)$ in the setup phase and t_o in the online phase), and

$$|\Pr[\mathcal{G}_2^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_1^{\mathcal{A}} = 1]| \leq \epsilon_{ZK}.$$

$\mathcal{G}_3$: $\mathcal{G}_3$ is identical to $\mathcal{G}_2$, except that $\mathbf{C}_2^*$ is computed as $\mathbf{Z}_2^*$ (instead of $\mathbf{Z}_2^* \cdot \mathbf{M}_b$).

We upper bound $|\Pr[\mathcal{G}_3^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_2^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{DRSW}$ to the decisional T-RSW problem. The reduction works because $\mathbf{R}_2$ is not used in DEC, so $\mathcal{R}_{DRSW}$ runs the code of $\mathcal{G}_3$, except that it uses the group element from the decisional T-RSW challenger as part of the challenge ciphertext. Concretely, $\mathcal{R}_{DRSW}$ works as follows:

– Preprocessing phase: $\mathcal{R}_{DRSW}$, on input N, chooses $g \leftarrow \mathbb{QR}_N$ and computes $\mathbf{H} := \mathsf{RepSqr}(g, N, 2^T)$; chooses $r_1^* \leftarrow \mathbb{Z}_{N^2}$ and computes $\mathbf{R}_1^* := g^{r_1^*}$ and $\mathbf{H}_1^* := \mathbf{H}^{r_1^*}$ (note that $\mathbf{R}_1^*$ and $\mathbf{H}_1^*$ can be computed in parallel); runs $(crs, td) \leftarrow \mathsf{SimGen}(1^\kappa)$. Then $\mathcal{R}_{DRSW}$ runs $\mathcal{A}(N, g, crs)$. $\mathcal{R}_{DRSW}$ answers $\mathcal{A}$'s DEC queries as described in $\mathcal{G}_1$.
– Online phase: When $\mathcal{A}$ makes its challenge query on $(\mathbf{M}_0, \mathbf{M}_1)$, $\mathcal{R}_{DRSW}$ asks for $(g^*, \mathbf{X}^*)$ from the decisional RSW challenger, chooses $b \leftarrow \{0,1\}$, computes $\mathbf{C}_1^* := \mathbf{H}_1^* \cdot \mathbf{M}_b$ and $\pi^* \leftarrow \mathsf{SimProve}((\mathbf{R}_1^*, g^*, \mathbf{C}_1^*, \mathbf{X}^*), td)$, and returns $(\mathbf{R}_1^*, g^*, \mathbf{C}_1^*, \mathbf{X}^*, \pi^*)$. $\mathcal{R}$ answers $\mathcal{A}$'s DEC queries just as in preprocessing.
– Output: On $\mathcal{A}$'s output bit b', $\mathcal{R}_{DRSW}$ outputs 1 if $b' = b$, and 0 otherwise.

Clearly, $\mathcal{R}_{DRSW}$ runs in time $t_p + T + t_{sgen} + \theta(\kappa)$ in the preprocessing phase, and time $t_o + t_{sprove}$ in the online phase, and

$$|\Pr[\mathcal{G}_3^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_2^{\mathcal{A}} = 1]| \leq \epsilon_{DRSW}.$$

$\mathcal{G}_4$: $\mathcal{G}_4$ is identical to $\mathcal{G}_3$, except that the DEC oracle uses $\mathbf{R}_2$ (instead of $\mathbf{R}_1$) to decrypt. That is, when $\mathcal{A}$ queries $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, let $r_2 := \log_g \mathbf{R}_2$, and compute $\mathbf{Z}_2 := \mathsf{RepSqr}(\mathbf{H}, N, r_2)$ and $\mathbf{M} := \dfrac{\mathbf{C}_2}{\mathbf{Z}_2}$.

$\mathcal{G}_4$ and $\mathcal{G}_3$ are identical unless $\mathcal{A}$ makes a query $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$ s.t. $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \neq \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}}$ but $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 1$ (in which case $\mathcal{A}$ receives $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}}$ in $\mathcal{G}_3$ and $\dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}}$ in $\mathcal{G}_4$; in all other cases $\mathcal{A}$ receives either $\perp$ in both games, or $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} = \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}}$ in both games). Denote this event Fake. We upper bound $\Pr[\mathsf{Fake}]$ by constructing a reduction $\mathcal{R}_{SS}$ to the simulation soundness property of NIZK:

– Setup: $\mathcal{R}_{SS}$, on input crs, chooses $g \leftarrow \mathbb{QR}_N$ and computes $\mathbf{H} := \mathsf{RepSqr}(g, N, 2^T)$; chooses $r_1^*, r_2^* \leftarrow \mathbb{Z}_{N^2}$ and computes

$$\mathbf{R}_1^* := \mathsf{RepSqr}(g, N, r_1^*), \mathbf{R}_2^* := \mathsf{RepSqr}(g, N, r_2^*),$$

$$\mathbf{Z}_1^* := \mathsf{RepSqr}(\mathbf{H}, N, r_1), \mathbf{Z}_2^* := \mathsf{RepSqr}(\mathbf{H}, N, r_2)$$

(note that $\mathbf{R}_1^*$, $\mathbf{R}_2^*$, $\mathbf{Z}_1^*$, $\mathbf{Z}_2^*$ can be computed in parallel). Then $\mathcal{R}_{SS}$ runs $\mathcal{A}(N, g, crs)$.
On $\mathcal{A}$'s query $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, $\mathcal{R}_{SS}$ computes $r_1 = \log_g \mathbf{R}_1$ and $r_2 = \log_g \mathbf{R}_2$ according to the algebraic coefficients $\mathcal{A}$ outputs, $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{H}, N, r_1)$, and $\mathbf{Z}_2 := \mathsf{RepSqr}(\mathbf{H}, N, r_2)$. If $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 0$, then $\mathcal{R}_{SS}$ returns $\perp$; otherwise $\mathcal{R}_{SS}$ checks if $\dfrac{\mathbf{C}_1}{\mathbf{Z}_1} = \dfrac{\mathbf{C}_2}{\mathbf{Z}_2}$, and if so, it returns $\dfrac{\mathbf{C}_1}{\mathbf{Z}_1}$, otherwise it outputs $((\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi)$ to its challenger (and halts).
– Online phase: When $\mathcal{A}$ makes its challenge query on $(\mathbf{M}_0, \mathbf{M}_1)$, $\mathcal{R}_{SS}$ chooses $b \leftarrow \{0,1\}$ and computes

$$\mathbf{C}_1^* := \mathbf{Z}_1^* \cdot \mathbf{M}_b, \mathbf{C}_2^* := \mathbf{Z}_2^*,$$

$$\pi^* \leftarrow \mathsf{SPROVE}(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*),$$

and outputs $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$. After that, $\mathcal{R}_{SS}$ answers $\mathcal{A}$'s DEC queries just as in setup.

Clearly, $\mathcal{R}_{SS}$ runs in time at most $t_p + t_o + T + \theta(\kappa)$ (i.e., $t_p + T + \theta(\kappa)$ in the setup phase and t_o in the online phase). Up to the point that $\mathcal{R}_{SS}$ outputs, $\mathcal{R}_{SS}$ simulates $\mathcal{G}_3$ perfectly. If Fake happens, then $\mathcal{R}_{SS}$ outputs $((\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi)$ s.t. $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \neq \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}}$ but $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 1$, winning the simulation-soundness game. It follows that

$$|\Pr[\mathcal{G}_4^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_3^{\mathcal{A}} = 1]| \leq \Pr[\mathsf{Fake}] \leq \Pr[\mathcal{R}_{SS} \text{ wins}] \leq \epsilon_{SS}.$$

$\mathcal{G}_5$: $\mathcal{G}_5$ is identical to $\mathcal{G}_4$, except that $\mathbf{C}_1^*$ is computed as $\mathbf{Z}_1^*$ (instead of $\mathbf{Z}_1^* \cdot \mathbf{M}_b$).

This game hop is symmetric to the one from $\mathcal{G}_2$ to $\mathcal{G}_3$; the reduction works because $\mathbf{R}_1$ is not used in DEC. We have that

$$|\Pr[\mathcal{G}_5^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_4^{\mathcal{A}} = 1]| \leq \epsilon_{DRSW}.$$

Furthermore, since b is independent of $\mathcal{A}$'s view in $\mathcal{G}_5$, we have that

$$\Pr[\mathcal{G}_5^{\mathcal{A}} = 1] = \frac{1}{2}.$$

Summing up all results above, we conclude that

$$\Pr\left[\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}^{\mathcal{A}} = 1\right] \leq \frac{1}{2} + \epsilon_{ZK} + \epsilon_{SS} + 2\epsilon_{DRSW},$$

which completes the proof.

7 Non-Malleable Timed-Commitments

In this section, we show how our notion of CCA-secure TRPKE (for a wide range of schemes) implies a very strong form of non-malleability for timed commitments as introduced by Boneh and Naor [6]. Timed commitments allow to commit to message m via a commitment C in such a way that C hides m up to some time T, yet the verifier can be sure that if the opening to C is not provided at that time, C can be forced open (with some computational effort). Timed commitments have several interesting applications in interactive protocols where the adversary could usually bias the outcome of the protocol by observing the opening to honest parties' commitments and then choosing whether or not to provide the openings to its own commitments. Boneh and Naor gave a (somewhat) informal description of the syntax of *interactive* timed-commitments and provided some specific constructions for them. In the following section, we follow their work and begin by introducing the syntax of a *non-interactive timed commitment scheme*. We then give appropriate security definitions for hiding and binding properties in the timed setting.

Definition 9 (Non-Interactive Timed-Commitment Scheme [6]). *A* $(t_{cm}, t_{cv}, t_{dv}, t_{fo})$-non-interactive timed-commitment scheme (NITIC) *is a tuple of algorithms* $\mathsf{TC} = (\mathsf{PGen}, \mathsf{Com}, \mathsf{ComVrfy}, \mathsf{DecomVrfy}, \mathsf{FDecom})$ *with the following behavior:*

- *The randomized* parameter generation algorithm PGen *takes as input the security parameter* 1^κ *and outputs a common reference string* crs.
- *The randomized* commit algorithm Com *takes as input a string* crs *and a message* m. *It outputs a commitment* C, *and proofs* $\pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}}$ *in time at most* t_{cm}.

- *The deterministic* commitment verification algorithm $\mathsf{ComVrfy}$ *takes as input a string* crs*, a commitment* C*, and a proof* π_{Com}*. It outputs* 1 *(accept) or* 0 *(reject) in time at most* t_{cv}*.*
- *The deterministic* decommitment verification algorithm $\mathsf{DecomVrfy}$ *takes as input a string* crs*, a commitment* C*, a message* m*, and a proof* π_{Decom}*. It outputs* 1 *(accept) or* 0 *(reject) in time at most* t_{dv}*.*
- *The deterministic* forced decommit algorithm FDecom *takes as input a string* crs *and a commitment* C*. It outputs a message* m *or* $\perp$ *in time at least* t_{fo}*.*

We briefly give an intuition of how a NITIC works. To commit to message m, the committer runs Com to get C, π_{Com} and π_{Decom}, and sends C and π_{Com} to the verifier. The verifier can (efficiently) run $\mathsf{ComVrfy}$ to check that C can be forcefully decommitted (if need be). To decommit, the committer sends m and π_{Decom} to the verifier. The verifier can (efficiently) run $\mathsf{DecomVrfy}$ to verify the opening m. If the committer refuses to reveal m, C can instead be opened (inefficiently) by running FDecom. We are mostly interested (similar to the TRPKE setting) in schemes s.t. $t_{cv} < t_{fo}$ and $t_{dv} < t_{fo}$, i.e., commitment verification and decommitment verification can always be done faster than forced decommit. *Correctness* of a NITIC informally captures that honestly formed commitments should verify correctly and that the forced decommit algorithm should produce the same value m that was used to compute the commitment.

Definition 10. *A NITIC scheme* $\mathsf{TC} := (\mathsf{PGen}, \mathsf{Com}, \mathsf{ComVrfy}, \mathsf{DecomVrfy}, \mathsf{FDecom})$ *is* perfectly correct *if for all* $m \in \{0,1\}^*$*, all* $crs \in \{\mathsf{PGen}(1^\kappa)\}$*, and all* $(C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}}) \in \{\mathsf{Com}(crs, m)\}$*, it holds that* $\mathsf{ComVrfy}(crs, C, \pi_{\mathsf{Com}}) = \mathsf{DecomVrfy}(crs, C, m, \pi_{\mathsf{Decom}}) = 1$ *and* $\mathsf{FDecom}(crs, C) = m$*.*

We next introduce the security game $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$ associated with the hiding property of a NITIC scheme. To capture the non-malleability property, we give the adversary access to an oracle that provides the (forced) openings to commitments of the adversary's choice. Our notion is directly inspired by the CCA-security notion for commitments given by Canetti et al. [8]. In the timed setting, the motivation behind providing the adversary with such an oracle is that parties may be running machines that can force open commitments at different speeds. As such, the adversary (as part of the higher-level protocol) could trick the committer into (quickly) forcing open commitments of the adversary's choice. This, in turn, may help the adversary to break the hiding property of the timed-commitment provided by the challenger prematurely.

$\underline{\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}}}$:

- **Setup:** $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$ samples $crs \leftarrow \mathsf{PGen}(1^\kappa)$ and a bit $b \leftarrow \{0,1\}$.
- **Preprocessing Phase:** $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$ runs $\mathcal{A}$ on input crs. In this phase, $\mathcal{A}$ is given access to a *decommit oracle* DEC, which on input C, returns $\mathsf{FDecom}(crs, C)$ in time t_{dc}.
- **Challenge Query:** When $\mathcal{A}$ outputs (m_0, m_1), $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$ computes $(C_b, \pi_b, \star) \leftarrow \mathsf{Com}(crs, m_b)$ and returns the challenge (C_b, π_b).
- **Online Phase:** After receiving (C_b, π_b), $\mathcal{A}$ continues to have access to DEC, except that $\mathcal{A}$ may not query DEC on C_b.
- **Output Determination:** When $\mathcal{A}$ outputs a bit b', $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$ outputs 1 if $b' = b$, and 0 otherwise.

Definition 11 (CCA Security for NITIC). *A NITIC scheme* TC *is* (ϵ, t_p, t_o)-CCA-secure *if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and in time* t_o *in the online phase of* $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$*,*

$$\Pr\left[\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

The binding and soundness properties are deferred to Appendix E.

7.1 Construction from TRPKE

We next show how a NITIC can be constructed from a TRPKE scheme. Our simple construction is presented in Fig. 2. For an efficient construction, we require that proofs for the relation R can be verified (very) efficiently, i.e., verifying takes less time than a forced decommit. This is satisfied when instantiating the TRPKE scheme with our construction from the previous section, where this relation can be expressed via an arithmetic circuit. Correctness of this scheme follows immediately

Let $\mathsf{TRPKE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ be a (t_e, t_{fd}, t_{sd})-TRPKE scheme, $\mathsf{NIZK}_{\mathsf{Com}} = (\mathsf{GenZK}_{\mathsf{Com}}, \mathsf{Prove}_{\mathsf{Com}}, \mathsf{Vrfy}_{\mathsf{Com}}, \mathsf{SimGen}_{\mathsf{Com}}, \mathsf{SimProve}_{\mathsf{Com}})$ be a $(t_{cp}, t_{cv}, t_{csgen}, t_{csp})$-NIZK for relation

$$R_{\mathsf{Com}} = \{(c, (m, r)) | c = \mathsf{Enc}(pk, m; r)\},$$

and $\mathsf{NIZK}_{\mathsf{Decom}} = (\mathsf{GenZK}_{\mathsf{Decom}}, \mathsf{Prove}_{\mathsf{Decom}}, \mathsf{Vrfy}_{,\mathsf{Decom}}, \mathsf{SimGen}_{\mathsf{Decom}}, \mathsf{SimProve}_{\mathsf{Decom}})$ be a $(t_{dp}, t_{dv}, t_{dsgen}, t_{dsp})$-NIZK for relation

$$R_{\mathsf{Decom}} = \{((c, m), r) | c = \mathsf{Enc}(pk, m; r)\}.$$

- $\mathsf{PGen}(1^\kappa)$: Run $par \leftarrow \mathsf{PGen}(1^\kappa)$ and $(sk, pk) \leftarrow \mathsf{KGen}(par)$, $crs_{\mathsf{Com}} \leftarrow \mathsf{GenZK}_{\mathsf{Com}}(1^\kappa)$, $crs_{\mathsf{Decom}} \leftarrow \mathsf{GenZK}_{\mathsf{Decom}}(1^\kappa)$, and output $crs := (pk, crs_{\mathsf{Com}}, crs_{\mathsf{Decom}})$.
- $\mathsf{Com}((pk, crs_{\mathsf{Com}}, crs_{\mathsf{Decom}}), m)$: Choose $r \leftarrow R$, compute $c := \mathsf{Enc}(pk, m; r)$, $\pi_{\mathsf{Com}} \leftarrow \mathsf{Prove}(crs_{\mathsf{Com}}, c, (m, r))$, $\pi_{\mathsf{Decom}} \leftarrow \mathsf{Prove}(crs_{\mathsf{Decom}}, (c, m), r)$, and output $(c, \pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}})$.
- $\mathsf{ComVrfy}((pk, crs_{\mathsf{Com}}, crs_{\mathsf{Decom}}), c, \pi_{\mathsf{Com}})$: Output $\mathsf{Vrfy}_{\mathsf{Com}}(crs_{\mathsf{Com}}, c, \pi_{\mathsf{Com}})$.
- $\mathsf{DecomVrfy}((pk, crs_{\mathsf{Com}}, crs_{\mathsf{Decom}}), c, m, \pi_{\mathsf{Decom}})$: Output $\mathsf{Vrfy}_{\mathsf{Decom}}(crs_{\mathsf{Decom}}, (c, m), \pi_{\mathsf{Decom}})$.
- $\mathsf{FDecom}((pk, crs_{\mathsf{Com}}, crs_{\mathsf{Decom}}), c)$: Output $\mathsf{Dec}_s(pk, c)$.

Fig. 2. A $(t_e + t_{cp} + t_{dp}, t_{cv}, t_{dv}, t_{sd})$-NITIC scheme from TRPKE

from correctness of the underlying TRPKE and NIZK schemes; we next show its CCA security.

Theorem 5. *Suppose that*

- TRPKE *is* $(\epsilon_{TRPKE}, t_p + t_{csgen}, t_{csp})$*-CCA secure, and*
- $\mathsf{NIZK}_{\mathsf{Com}}$ *is* $(\epsilon_{ZK}, t_p + t_o + t_e)$*-zero-knowledge.*

Then the NITICS scheme in Fig. 2 is $(\epsilon_{ZK} + \epsilon_{CCA}, t_p, t_o)$*-CCA-secure.*

The proof is rather straightforward and is deferred to Appendix B.

References

1. M. Bellare and S. Goldwasser. Verifiable partial key escrow. In R. Graveman, P. A. Janson, C. Neuman, and L. Gong, editors, *ACM CCS 97: 4th Conference on Computer and Communications Security*, pages 78–91, Zurich, Switzerland, Apr. 1–4, 1997. ACM Press.
2. N. Bitansky, S. Garg, H. Lin, R. Pass, and S. Telang. Succinct randomized encodings and their applications. In R. A. Servedio and R. Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press.

3. N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In M. Sudan, editor, *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356, Cambridge, MA, USA, Jan. 14–16, 2016. Association for Computing Machinery.
4. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.
5. D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://eprint.iacr.org/2018/712`.
6. D. Boneh and M. Naor. Timed commitments. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, Aug. 20–24, 2000. Springer, Heidelberg, Germany.
7. D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In K. Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 59–71, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.
8. R. Canetti, H. Lin, and R. Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st Annual Symposium on Foundations of Computer Science*, pages 541–550, Las Vegas, NV, USA, Oct. 23–26, 2010. IEEE Computer Society Press.
9. J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In S. Qing, W. Mao, J. López, and G. Wang, editors, *ICICS 05: 7th International Conference on Information and Communication Security*, volume 3783 of *Lecture Notes in Computer Science*, pages 291–303, Beijing, China, Dec. 10–13, 2005. Springer, Heidelberg, Germany.
10. G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In J. Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 74–89, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.
11. C. Dwork and M. Naor. Zaps and their applications. In *41st Annual Symposium on Foundations of Computer Science*, pages 283–293, Redondo Beach, CA, USA, Nov. 12–14, 2000. IEEE Computer Society Press.
12. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.
13. J. Groth and M. Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612, Santa Barbara, CA, USA, Aug. 20–24, 2017. Springer, Heidelberg, Germany.
14. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.
15. A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. Papamanthou, R. Pass, a. shelat, and E. Shi. How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, Report 2015/1093, 2015. `http://eprint.iacr.org/2015/1093`.
16. H. Lin, R. Pass, and P. Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In C. Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 576–587, Berkeley, CA, USA, Oct. 15–17, 2017. IEEE Computer Society Press.
17. H. Lipmaa. Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612, 2019. `https://eprint.iacr.org/2019/612`.
18. J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi. How to build time-lock encryption. In *Designs, Codes and Cryptogrpahy*, 2018.
19. M. Mahmoody, T. Moran, and S. P. Vadhan. Time-lock puzzles in the random oracle model. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50, Santa Barbara, CA, USA, Aug. 14–18, 2011. Springer, Heidelberg, Germany.
20. G. Malavolta and S. A. K. Thyagarajan. Homomorphic time-lock puzzles and applications. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649, Santa Barbara, CA, USA, Aug. 18–22, 2019. Springer, Heidelberg, Germany.
21. U. M. Maurer. Abstract models of computation in cryptography (invited paper). In N. P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12, Cirencester, UK, Dec. 19–21, 2005. Springer, Heidelberg, Germany.
22. T. C. May. Timed-release crypto. Technical report, 1993.

23. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, Baltimore, MD, USA, May 14–16, 1990. ACM Press.
24. P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In B. K. Roy, editor, *Advances in Cryptology – ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 1–20, Chennai, India, Dec. 4–8, 2005. Springer, Heidelberg, Germany.
25. K. Pietrzak. Simple verifiable delay functions. In A. Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, Jan. 10–12, 2019. LIPIcs.
26. R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. Technical report, MIT, 1996.
27. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.
28. B. Wesolowski. Efficient verifiable delay functions. In Y. Ishai and V. Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

A Additional Definitions

A.1 Authenticated Encryption

Definition 12 (Authenticated Encryption Scheme). *A* (t_e, t_d)-authenticated encryption scheme *is a tuple of algorithms* $\mathsf{AE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *with the following behavior:*

- *The randomized* parameter generation *algorithm* PGen *takes as input the security parameter* 1^κ *and outputs public parameters par.*
- *The randomized* key generation algorithm KGen *takes as input public parameters par and outputs a key k.*
- *The randomized* encryption algorithm Enc *takes as input a key k and a message* $m \in \{0,1\}^*$*. It outputs a ciphertext* $c \in \{0,1\}^*$ *and runs in time at most* t_e *for all m.*
- *The deterministic* decryption algorithm Dec *takes as input a key k and a ciphertext* $c \in \{0,1\}^*$*. It outputs a message* $m \in \{0,1\}^*$ *or* $\perp$ *and runs in time at most* t_d *for all c.*

We say that AE has *perfect correctness* if for all $m \in \{0,1\}^*$, all $par \in \{\mathsf{PGen}(1^\kappa)\}$, all $k \in \{\mathsf{KGen}(par)\}$, and all $c \in \{\mathsf{Enc}(k,m)\}$, it holds that $m = \mathsf{Dec}(k,c)$.

In this work, we are interested in two different security properties of an authenticated encryption scheme AE which we now define via the following two games. We remark that our game for CPA security differs slightly from the literature in that we explicitly split the game into two phases: a *preprocessing phase* that comes before the challenge query and an *online phase* that starts as soon as the adversary $\mathcal{A}$ has made the challenge query.

$\underline{\textbf{IND-CPA}^{\mathcal{A}}_{\mathsf{AE}}}$:

- **Setup: IND-CPA**$_{\mathsf{AE}}$ samples parameters *par* via $par \leftarrow \mathsf{PGen}(1^\kappa)$ and a bit $b \leftarrow \{0,1\}$. It then samples a key k as $k \leftarrow \mathsf{KGen}(par)$.
- **Preprocessing Phase: IND-CPA**$_{\mathsf{AE}}$ runs $\mathcal{A}$. In this phase, $\mathcal{A}$ is given access to an *encryption oracle* ENC, which on input m, returns a ciphertext $c \leftarrow \mathsf{Enc}(k,m)$.
- **Challenge Query:** When $\mathcal{A}$ outputs (m_0, m_1), **IND-CPA**$_{\mathsf{AE}}$ returns the challenge ciphertext $c_b \leftarrow \mathsf{Enc}(k, m_b)$.
- **Online Phase:** After receiving c_b, $\mathcal{A}$ continues to have access to ENC.
- **Output Determination:** When $\mathcal{A}$ outputs a bit b', **IND-CPA**$_{\mathsf{AE}}$ outputs 1 if $b' = b$, and 0 otherwise.

Definition 13 (CPA Security for Symmetric Encryption). *An (authenticated) symmetric encryption scheme* AE *is* (ϵ, t_p, t_o)-CPA secure *if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase of* $\mathbf{IND\text{-}CPA}_{\mathsf{AE}}$*,*

$$\Pr\left[\mathbf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{AE}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

We next formalize the integrity of ciphertexts property of an authenticated encryption scheme AE via the following game.

$\underline{\mathbf{INT\text{-}CTXT}^{\mathcal{A}}_{\mathsf{AE}}}$:

- **Setup:** $\mathbf{INT\text{-}CTXT}_{\mathsf{AE}}$ samples public parameters *par* via $par \leftarrow \mathsf{PGen}(1^\kappa)$. It then samples a key k as $k \leftarrow \mathsf{KGen}(par)$ and initializes set $\mathcal{S} := \emptyset$.
- **Encryption Queries:** $\mathbf{INT\text{-}CTXT}_{\mathsf{AE}}$ runs $\mathcal{A}$. $\mathcal{A}$ is given access to an *encryption oracle* ENC, which on input m, returns a ciphertext $c \leftarrow \mathsf{Enc}(k, m)$ and sets $\mathcal{S} := \mathcal{S} \cup \{c\}$.
- **Output Determination:** When $\mathcal{A}$ outputs a ciphertext c, $\mathbf{INT\text{-}CTXT}_{\mathsf{AE}}$ outputs 1 if $c \notin \mathcal{S}$ and $\mathsf{Dec}(k, c) \neq \perp$. It outputs 0 otherwise.

Definition 14 (INT-CTXT Security for Authenticated Encryption). *An authenticated symmetric encryption scheme* AE *is* (ϵ, t)*-INT-CTXT secure if for all adversaries* $\mathcal{A}$ *running in time* t *in* $\mathbf{INT\text{-}CTXT}_{\mathsf{AE}}$*,*

$$\Pr\left[\mathbf{INT\text{-}CTXT}^{\mathcal{A}}_{\mathsf{AE}} = 1\right] \leq \epsilon.$$

A.2 Non-Interactive Zero-Knowledge

We recall the notion of a non-interactive zero-knowledge proof system, defined as follows.

Definition 15 (Non-Interactive Zero-Knowledge Proof System). *Let* $\mathcal{L}_R$ *be a language in NP defined by relation* R*. A* $(t_{prove}, t_{vrfy}, t_{sgen}, t_{sprove})$-non-interactive zero-knowledge proof (NIZK) system (for relation R) *is a tuple of algorithms* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *with the following behavior:*

- *The randomized* parameter generation algorithm GenZK *takes as input the security parameter* 1^κ *and outputs a common reference string* crs*.*
- *The randomized* prover algorithm Prove *takes as input a string* crs*, an instance* x*, and a witness* w*. It outputs a proof* π *and runs in time at most* t_{prove} *for all* crs*,* x *and* w*.*
- *The deterministic* verifier algorithm Vrfy *takes as input a string* crs*, an instance* x*, and a proof* π*. It outputs* 1 *(accept) or* 0 *(reject) and runs in time at most* t_{vrfy} *for all* crs*,* x *and* π*.*
- *The randomized* simulation parameter generation algorithm SimGen *takes as input the security parameter* 1^κ*. It outputs a common reference string* crs *and a trapdoor* td *and runs in time at most* t_{sgen}*.*
- *The randomized* simulation prover algorithm $\mathsf{SimProve}$ *takes as input an instance* x *and a trapdoor* td*. It outputs a proof* π *and runs in time at most* t_{sprove} *for all* x *and* $td \in \{\mathsf{SimGen}(1^\kappa)\}$*.*

We require perfect completeness*: For all* $crs \in \{\mathsf{GenZK}(1^\kappa)\}$*, all* $(x, w) \in R$*, and all* $\pi \in \{\mathsf{Prove}(crs, x, w)\}$*, it holds that* $\mathsf{Vrfy}(crs, x, \pi) = 1$*.*

We next define zero-knowledge and soundness properties of a NIZK.

Definition 16 (Zero-Knowledge). *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation R. The* zero-knowledge property of NIZK *is defined via the following game* $\mathbf{ZK}_{\mathsf{NIZK}}$*:*

- ***Setup:*** $\mathbf{ZK}_{\mathsf{NIZK}}$ *samples crs_0, crs_1 via $crs_0 \leftarrow \mathsf{GenZK}(1^\kappa)$, $crs_1 \leftarrow \mathsf{SimGen}(1^\kappa)$, and a bit $b \leftarrow \{0,1\}$. It then runs $\mathcal{A}$ on input crs_b.*
- ***Online Phase:*** *During this phase of the game, $\mathcal{A}$ is given access to a* prover oracle PROVE*. On input (x, w),* PROVE *returns $\perp$ if $(x, w) \notin R$. Otherwise, it generates $\pi_0 \leftarrow \mathsf{Prove}(crs_0, x, w), \pi_1 \leftarrow \mathsf{SimProve}(crs_1, x, w)$ and returns π_b.*
- ***Output Determination:*** *When $\mathcal{A}$ returns b',* $\mathbf{ZK}_{\mathsf{NIZK}}$ *outputs 1 if $b' = b$, and 0 otherwise.*

We say that NIZK *is* (ϵ, t)-zero-knowledge *if for all adversaries $\mathcal{A}$ running in time t,*

$$\Pr\left[\mathbf{ZK}^{\mathcal{A}}_{\mathsf{NIZK}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

Definition 17 (Soundness). *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation R. The* soundness of NIZK *is defined via the following game* $\mathbf{SND}_{\mathsf{NIZK}}$*:*

- ***Setup:*** $\mathbf{SND}_{\mathsf{NIZK}}$ *samples crs via $crs \leftarrow \mathsf{GenZK}(1^\kappa)$. It runs $\mathcal{A}$ on input crs.*
- ***Output Determination:*** *When $\mathcal{A}$ returns (x, π),* $\mathbf{SND}_{\mathsf{NIZK}}$ *outputs 1 if $\mathsf{Vrfy}(crs, x, \pi) = 1$ and $x \notin \mathcal{L}_R$. It outputs 0 otherwise.*

We say that NIZK *is* (ϵ, t)-sound *if for all adversaries $\mathcal{A}$ running in time t,*

$$\Pr\left[\mathbf{SND}^{\mathcal{A}}_{\mathsf{NIZK}} = 1\right] \leq \epsilon.$$

In our applications we also need the stronger notion of simulation soundness, which says that the adversary cannot produce a fake proof even if it has oracle access to the simulated prover algorithm.

Definition 18 (Simulation Soundness). *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation R. The* simulation soundness of NIZK *is defined via the following game* $\mathbf{SIMSND}_{\mathsf{NIZK}}$*:*

- ***Setup:*** $\mathbf{SIMSND}_{\mathsf{NIZK}}$ *samples crs via $crs \leftarrow \mathsf{SimGen}(1^\kappa)$ and initializes $\mathcal{Q} := \emptyset$. It runs $\mathcal{A}$ on input crs.*
- ***Online Phase:*** *During this phase of the game, $\mathcal{A}$ is given access to a* simulated prover oracle SPROVE*. On input (x, w),* SPROVE *generates $\pi \leftarrow \mathsf{SimProve}(x, t)$, sets $\mathcal{Q} := \mathcal{Q} \cup \{x\}$, and returns π.*
- ***Output Determination:*** *When $\mathcal{A}$ returns (x, π),* $\mathbf{SIMSND}_{\mathsf{NIZK}}$ *outputs 1 if $x \notin \mathcal{Q}$, $\mathsf{Vrfy}(crs, x, \pi) = 1$, and $x \notin \mathcal{L}_R$.*

We say that $\mathbf{ZK}_{\mathsf{NIZK}}$ *is* (ϵ, t)-simulation sound *if for all adversaries $\mathcal{A}$ running in time t,*

$$\Pr\left[\mathbf{SND}^{\mathcal{A}}_{\mathsf{NIZK}} = 1\right] \leq \epsilon.$$

For integers N s.t. $N = pq$ for primes p and q, let C be an arithmetic circuit over $\mathbb{Z}_N$, and let SAT_C denote the set of all $(x, w) \in \{0,1\}^*$ s.t. w is a satisfying assignment to C when C's wires are fixed according to x. The works of Groth and Maller [13] as well as Lipmaa [17] show NIZK constructions for SAT_C which have soundness and simulation soundness (with suitable parameters), perfect zero-knowledge, perfect correctness and are such that for all $crs \in \{\mathsf{GenZK}(1^\kappa)\}, (crs', td) \in \{\mathsf{SimGen}(1^\kappa)\}$, all $(x, w) \in \mathsf{SAT}_C$ and all $x' \in \{0,1\}^*$:

– For all $\pi \in \{\mathsf{Prove}(crs, x, w)\}$, Vrfy runs within time $O(|x|)$ on input (crs, x, π).
– For all $\pi' \in \{\mathsf{SimProve}(x', td)\}$, Vrfy runs within time $O(|x'|)$ on input (crs', x', π').
– On input (x', td), $\mathsf{SimProve}$ runs in time $O(|x'|)$.[6]

Crucial to our application is that both Vrfy and $\mathsf{SimProve}$ run in a fast manner, i.e., linear in the scale of the input instance.

B Proof of Theorem 5

Proof. Let $\mathcal{A}$ be an adversary in the CCA Security Game for this NITICS scheme, $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$, with preprocessing time t_p and online time t_o. Suppose $\mathcal{A}$'s challenge is (c^*, π^*). The proof goes by a sequence of games, which we describe next.

$\mathcal{G}_0$: $\mathcal{G}_0$ is the original CCA Security Game, $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$.

$\mathcal{G}_1$: $\mathcal{G}_1$ is identical to $\mathcal{G}_0$, except that crs_{Com} and π^* are simulated. That is, in the setup phase run $(crs_{\mathsf{Com}}, td) \leftarrow \mathsf{SimGen}_{\mathsf{Com}}(1^\kappa)$, and in the challenge compute $\pi^* \leftarrow \mathsf{SimProve}_{\mathsf{Com}}(c^*, td)$.

We upper bound $|\Pr[\mathcal{G}_1^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_0^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{ZK}$ to the zero-knowledge property of $\mathsf{NIZK}_{\mathsf{Com}}$. $\mathcal{R}_{ZK}$ runs the code of $\mathcal{G}_1$, except that it publishes the CRS from the zero-knowledge challenger, and uses the zero-knowledge proof from the zero-knowledge challenger as part of the challenge ciphertext; also, $\mathcal{R}_{ZK}$ simulates the decommit oracle DEC by running the fast decryption algorithm. Concretely, $\mathcal{R}_{ZK}$ works as follows:

– Setup: $\mathcal{R}_{ZK}$, on input crs^*, runs $par \leftarrow \mathsf{PGen}(1^\kappa)$, $(sk, pk) \leftarrow \mathsf{KGen}(par)$ and $crs_{\mathsf{Decom}} \leftarrow \mathsf{GenZK}_{\mathsf{Decom}}(1^\kappa)$, sets $crs := (pk, crs^*, crs_{\mathsf{Decom}})$, and runs $\mathcal{A}(crs)$.
  On $\mathcal{A}$'s query $\mathsf{DEC}(c)$, $\mathcal{R}_{ZK}$ returns $\mathsf{Dec}_s(sk, c)$.
– Online phase: When $\mathcal{A}$ makes its challenge query on (m_0, m_1), $\mathcal{R}_{ZK}$ chooses $b \leftarrow \{0, 1\}$, computes $c^* \leftarrow \mathsf{Enc}(pk, m_b)$ and $\pi^* \leftarrow \mathsf{PROVE}(c^*, m_b)$, and outputs (c, π^*). After that, $\mathcal{R}$ answers $\mathcal{A}$'s DEC queries just as in setup.
– Output: On $\mathcal{A}$'s output bit b', $\mathcal{R}_{ZK}$ outputs 1 if $b' = b$, and 0 otherwise.

Clearly, $\mathcal{R}_{ZK}$ runs in time $t_p + t_o + t_e$ (i.e., time t_p in the setup phase and $t_o + t_e$ in the online phase), and

$$|\Pr[\mathcal{G}_1^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_0^{\mathcal{A}} = 1]| \leq \epsilon_{ZK}.$$

Now we analyze $\mathcal{A}$'s advantage in $\mathcal{G}_1$. Since the challenge is (c, π) where $c = \mathsf{Enc}(pk, m; r)$ and π is simulated without knowledge of m or r, and DEC simply runs Dec_s, $\mathcal{A}$'s advantage can be upper bounded directly by the CCA security of TRPKE. Formally, we upper bound $\mathcal{A}$'s advantage by constructing a reduction $\mathcal{R}_{CCA}$ to the CCA security of TRPKE (where $\mathcal{R}_{CCA}$'s decryption oracle is denoted $\mathsf{DEC}_{\mathsf{TRPKE}}$):

– Preprocessing Phase: $\mathcal{R}_{CCA}$, on input pk, computes $(crs_{\mathsf{Com}}, td) \leftarrow \mathsf{SimGen}_{\mathsf{Com}}(1^\kappa)$, and runs $\mathcal{A}(crs_{\mathsf{Com}})$.
  On $\mathcal{A}$'s query $\mathsf{DEC}(c)$, $\mathcal{R}_{CCA}$ queries $\mathsf{DEC}_{\mathsf{TRPKE}}(c)$ and returns the result.
– Challenge Query: When $\mathcal{A}$ outputs (m_0, m_1), $\mathcal{R}_{CCA}$ makes its challenge query on (m_0, m_1), and on its challenge ciphertext c^*, $\mathcal{R}_{CCA}$ computes $\pi^* \leftarrow \mathsf{SimProve}_{\mathsf{Com}}(c^*, td)$ and sends (c^*, π^*) to $\mathcal{A}$. After that, $\mathcal{R}$ answers $\mathcal{A}$'s DEC queries just as in preprocessing phase.

[6] These construction work over $\mathbb{Z}_p$ for primes p only, but can translated to $\mathbb{Z}_N$, where N is composite, with constant overhead, as shown in [15].

– Output: When $\mathcal{A}$ outputs a bit b', $\mathcal{R}_{CCA}$ also outputs b'.

Clearly, $\mathcal{R}_{CCA}$ runs in time at most $t_p + t_{csgen}$ in the preprocessing phase, and time at most $t_o + t_{csp}$ in the online phase. $\mathcal{R}_{CCA}$ simulates $\mathcal{G}_1$ perfectly, and if $\mathcal{A}$ wins, then $\mathcal{R}_{CCA}$ wins. It follows that

$$\Pr[\mathcal{G}_1^{\mathcal{A}} = 1] = \Pr[\mathcal{R}_{CCA} \text{ wins}] \leq \frac{1}{2} + \epsilon_{CCA}.$$

Summing up all results above, we conclude that

$$\Pr\left[\mathbf{IND\text{-}CCA}_{\mathsf{TC}}^{\mathcal{A}} = 1\right] \leq \frac{1}{2} + \epsilon_{ZK} + \epsilon_{CCA},$$

which completes the proof.

C Definition for Time-Released Encryption with Fast Encryption

In the following, we define time-released encryption with an additional property called *fast encryption* that allows the holder of the secret key sk to encrypt much faster than what would be possible using only pk. As we will see, this requires not only a modification to the definition of TRPKE, but also to the security games, as it now makes sense to introduce an additional oracle ENC that allows to encrypt plaintexts *fast*.

Definition 19 (Time-Released Encryption With Fast Encryption). *A* $(t_{fe}, t_{se}, t_{fd}, t_{sd})$-time-released encryption scheme with fast encryption (TRFE) *is a tuple of algorithms* $\mathsf{TRFE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}_f, \mathsf{Enc}_s, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *where* $(\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}_s, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *is a* (t_{se}, t_{fd}, t_{sd})*-time-released public-key encryption (TRPKE) scheme and the randomized* fast encryption algorithm Enc_f *has the following behavior: it takes as input a secret key* sk *and a message* $m \in \{0,1\}^*$*. It outputs a ciphertext* $c \in \{0,1\}^*$ *and runs in time at most* t_{fe} *for all* m*.*

In addition to $t_{fd} < t_{sd}$, we require that $t_{fe} < t_{se}$ i.e., encrypting fast is faster than encrypting slow. We begin by adapting the definition of perfect correctness to account for the two different ways to encrypt:

Definition 20 (Perfect Correctness). *A TRFE scheme* $\mathsf{TRFE} = (\mathsf{PGen}, \mathsf{KGen}, \mathsf{Enc}_f, \mathsf{Enc}_s, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *satisfies* perfect correctness *if for all* $m \in \{0,1\}^*$*, all* $par \in \{\mathsf{PGen}(1^\kappa)\}$*, all* $(sk, pk) \in \{\mathsf{KGen}(par)\}$*, and all* $c \in \{\mathsf{Enc}_f(sk, m)\} \cup \{\mathsf{Enc}_s(pk, m)\}$*, it holds that* $\mathsf{Dec}_f(sk, c) = \mathsf{Dec}_s(pk, c) = m$*.*

We now define appropriate security notions for TRFE, this time giving two different notions which (as for TRPKE) resemble the standard indistinguishability security notions. An interesting feature of TRFE, however, is that it can be seen as lying in between public and secret key encryption, as it bears resemblance to some aspects of both of these primitives. Namely, as for public-key encryption, an adversary that has the public key of a TRFE scheme can compute ciphertexts of arbitrary messages using Enc_s. However, these ciphertexts are not useful to the adversary in an indistinguishability game if $t_{se} \approx t_{sd}$ (which is indeed the case for all of our proposed constructions), because by the time it can encrypt even a single message, it could have just as well decrypted the challenge ciphertext.

Therefore, as for the symmetric-key setting, our games offer an oracle ENC which the adversary can query for ciphertexts on arbitrary messages of its choice. The main point of ENC is that it replies to queries within time at most t_{fe} (by running the fast encryption algorithm), whereas the adversary's computation of a ciphertext via Enc_s would take time at least t_{se} (because the adversary does not know the secret key, so it can only run the slow encryption algorithm).

We describe the CPA Security Game $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$ for a TRFE scheme TRFE and an adversary $\mathcal{A}$.

$\underline{\mathbf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{TRFE}}}$:

- **Setup:** $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$ samples parameters par via $par \leftarrow \mathsf{PGen}(1^\kappa)$ and a bit $b \leftarrow \{0,1\}$. It then samples a pair of keys (sk, pk) as $(sk, pk) \leftarrow \mathsf{KGen}(par)$.
- **Preprocessing Phase:** $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$ runs $\mathcal{A}$ on input pk. In this phase, $\mathcal{A}$ is given access to an *encryption oracle* ENC, which on input m, returns a ciphertext $c \leftarrow \mathsf{Enc}_f(sk, m)$ *in time* t_{fe}.
- **Challenge Query:** When $\mathcal{A}$ outputs (m_0, m_1), $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$ returns the challenge ciphertext $c_b \leftarrow \mathsf{Enc}_f(sk, m_b)$.
- **Online Phase:** After receiving c_b, $\mathcal{A}$ continues to have access to ENC.
- **Output Determination:** When $\mathcal{A}$ outputs a bit b', $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$ returns 1 if $b' = b$, and 0 otherwise.

Definition 21 (CPA Security for TRFE). *A TRFE scheme* TRFE *is* (ϵ, t_p, t_o)-CPA-secure *(where* $t_o < t_{sd}$*) if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase of* $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$*,*

$$\Pr\left[\mathbf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{TRFE}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

We next consider the security notion of Indistinguishability under Chosen Ciphertext Attacks (CCA). Its security game $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$ is identical to $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$, except that it offers an oracle DEC which the adversary can query for plaintexts on arbitrary ciphertexts of its choice (except for the challenge ciphertext). DEC replies to queries within time at most t_{fd} (by running the fast decryption algorithm), similar to the encryption oracle in $\mathbf{IND\text{-}CPA}_{\mathsf{TRFE}}$.

$\underline{\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TRFE}}}$:

- **Setup:** $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$ samples parameters par via $par \leftarrow \mathsf{PGen}(1^\kappa)$ and a bit $b \leftarrow \{0,1\}$. It then samples a pair of keys (sk, pk) as $(sk, pk) \leftarrow \mathsf{KGen}(par)$.
- **Preprocessing Phase:** $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$ runs $\mathcal{A}$ on input pk. In this phase, $\mathcal{A}$ is given access to an *encryption oracle* ENC, which on input m, returns a ciphertext $c \leftarrow \mathsf{Enc}_f(sk, m)$ *in time* t_{fe}; as well as a *decryption oracle* DEC, which on input c, returns a plaintext $m := \mathsf{Dec}_f(sk, c)$ *in time* t_{fd}.
- **Challenge Query:** When $\mathcal{A}$ outputs (m_0, m_1), $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$ returns the challenge ciphertext $c_b \leftarrow \mathsf{Enc}_f(sk, m_b)$.
- **Online Phase:** After receiving c_b, $\mathcal{A}$ continues to have access to ENC and DEC, except that $\mathcal{A}$ may not query DEC on c_b.
- **Output Determination:** When $\mathcal{A}$ outputs a bit b', $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$ returns 1 if $b' = b$, and 0 otherwise.

Definition 22 (CCA Security for TRFE). *A TRFE scheme* TRFE *is* (ϵ, t_p, t_o)-CCA-secure *(where* $t_o < t_{sd}$*) if for all adversaries* $\mathcal{A}$ *running in time* t_p *in the preprocessing phase and* t_o *in the online phase of* $\mathbf{IND\text{-}CCA}_{\mathsf{TRFE}}$*,*

$$\Pr\left[\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TRFE}} = 1\right] \leq \frac{1}{2} + \epsilon.$$

D TRFE Constructions in the ROM

D.1 An CPA-Secure Construction

We construct a CPA-secure TRFE scheme over $\mathbb{QR}_N$ in Fig. 3, based on the hardness of the RSW problem and in the random oracle model. The key observation is that if $\phi = \phi(N)$ is known, then $H\left(\mathbf{R}^{2^T}\right)$ (where $\mathbf{R}$ is a random element in $\mathbb{QR}_N$ and H is modeled as a random oracle) can be computed fast; otherwise it is pseudorandom until a certain time t. Therefore, if we take any secret-key encryption scheme and replace the key with $H\left(\mathbf{R}^{2^T}\right)$, then we get a TLE scheme where the public and secret keys are N and (N, ϕ), respectively. Importantly, a fresh $\mathbf{R}$ is chosen every time an encryption is done, hence preventing any form of preprocessing attacks. In our construction we use one-time pad as the underlying secret-key encryption scheme, so the ciphertext is

$$\left(\mathbf{R}, H\left(\mathbf{R}^{2^T}\right) \oplus m\right).$$

Furthermore, since $\mathbf{R}$ is freshly chosen every time the adversary queries the encryption oracle (and generates an independent random "pad" every time), the encryption oracle is useless to the adversary. Thus, the TRFE scheme is CPA-secure.

Let ℓ be polynomial in κ, and $H : \mathbb{QR}_N \to \{0,1\}^\ell$ be a hash function (modeled as a random oracle).
The message space and the ciphertext space are $\{0,1\}^\ell$ and $\mathbb{QR}_N \times \{0,1\}^\ell$, respectively.

- $\mathsf{PGen}(1^\kappa)$: Run $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$, compute $\phi := \phi(N) = (p-1)(q-1)$ and output $par := (N, \phi)$.
- $\mathsf{KGen}(N, \phi)$: Output $(sk := (N, \phi), pk := N)$.
- $\mathsf{Enc}_f((N, \phi), m)$: Choose $\mathbf{R} \leftarrow \mathbb{QR}_N$, compute $z := [2^T \bmod \frac{\phi}{4}]$ and $\mathbf{Z} := \mathsf{RepSqr}(\mathbf{R}, N, z)$, and output $(\mathbf{R}, c := H(\mathbf{Z}) \oplus m)$.
- $\mathsf{Enc}_s(N, m)$: Choose $\mathbf{R} \leftarrow \mathbb{QR}_N$, compute $\mathbf{Z} := \mathsf{RepSqr}(\mathbf{R}, N, 2^T)$, and output $(\mathbf{R}, c := H(\mathbf{Z}) \oplus m)$.
- $\mathsf{Dec}_f((N, \phi), (\mathbf{R}, c))$: Compute $z := [2^T \bmod \frac{\phi}{4}]$ and $\mathbf{Z} := \mathsf{RepSqr}(\mathbf{R}, N, z)$, and output $m := H(\mathbf{Z}) \oplus c$.
- $\mathsf{Dec}_s(N, (\mathbf{R}, c))$: Compute $\mathbf{Z} := \mathsf{RepSqr}(\mathbf{R}, N, 2^T)$, and output $m := H(\mathbf{Z}) \oplus c$.

Fig. 3. An CPA-secure $(2\theta(\kappa), T + \theta(\kappa), 2\theta(\kappa), T)$-TLE scheme

Theorem 6. *Suppose that the* T*-RSW problem is* $(\epsilon_{RSW}, t_p + T + \theta(\kappa), t_o)$*-hard relative to* GenMod*. Then the TRFE scheme in Fig. 3 is* $(Q_H \cdot \epsilon_{RSW} + \frac{Q_E}{2^{\theta(\kappa)}}, t_p, t_o)$*-CPA-secure, where the adversary queries the encryption oracle* ENC *at most* Q_E *times and the random oracle* H *at most* Q_H *times.*

Proof. Let $\mathcal{A}$ be an adversary in the CPA Security Game for this TRFE scheme, $\mathbf{IND\text{-}CPA}_{\mathsf{TRPKE}}$, with preprocessing time t_p and online time t_o, querying the encryption oracle ENC at most Q_E times and the random oracle H at most Q_H times. Suppose $\mathcal{A}$'s challenge ciphertext is $(\mathbf{R}^*, c^*)$, and its i-th ENC query ($i = 1, \ldots, Q_E$) is answered with $(\mathbf{R}^{(i)}, c^{(i)})$. Let Repeat be the event that $\mathbf{R}^* = \mathbf{R}^{(i)}$ for some i, and Query be the event that $\mathcal{A}$ queries $H\left((\mathbf{R}^*)^{2^T}\right)$. Since $\mathbf{R}^*$ and all $\mathbf{R}^{(i)}$ are independently chosen at random from $\mathbb{QR}_N$, we have that

$$\Pr[\mathsf{Repeat}] \leq \frac{Q_E}{|\mathbb{QR}_N|} < \frac{Q_E}{2^{\theta(\kappa)}}.$$

We upper bound $\Pr[\mathsf{Query}]$ by constructing a reduction $\mathcal{R}$ to the T-RSW problem (where the inputs are N and g^*):

- Preprocessing phase: $\mathcal{R}$, on input N, chooses $j' \leftarrow \{1, \ldots, Q_H\}$ (a guess that $\mathcal{A}$ causes Query to happen for the first time when it makes the j'-th H query). For $i = 1, \ldots, Q_E$, $\mathcal{R}$ chooses $\mathbf{R}^{(i)} \leftarrow \mathbb{QR}_N$, computes $\mathbf{Z}^{(i)} := \mathsf{RepSqr}(\mathbf{R}^{(i)}, N, 2^T)$, and records $(\mathbf{R}^{(i)}, \mathbf{Z}^{(i)})$. (These Q_E operations can be done in parallel.) Then $\mathcal{R}$ runs $\mathcal{A}(N)$ and answers $\mathcal{A}$'s oracle queries as follows:
  - When $\mathcal{A}$ makes the i-th encryption oracle query $\mathsf{ENC}(m^{(i)})$, $\mathcal{R}$ finds its record $(\mathbf{R}^{(i)}, \mathbf{Z}^{(i)})$, sets $c^{(i)} := H(\mathbf{Z}^{(i)}) \oplus m^{(i)}$ ($\mathcal{R}$ chooses $H(\mathbf{Z}^{(i)}) \leftarrow \{0,1\}^\ell$ if it is undefined), and outputs $(\mathbf{R}^{(i)}, c^{(i)})$ to $\mathcal{A}$.
  - $\mathcal{R}$ answers $\mathcal{A}$'s H queries via lazy sampling.
- Challenge query: When $\mathcal{A}$ makes its challenge query, $\mathcal{R}$ asks for $\mathbf{R}^*$ from the RSW challenger. If $\mathbf{R}^* = \mathbf{R}^{(i)}$ for some i, then $\mathcal{R}$ outputs $\mathbf{Z}^{(i)}$ (and halts). Otherwise $\mathcal{R}$ chooses $c^* \leftarrow \{0,1\}^\ell$ and returns $(\mathbf{R}^*, c^*)$.
- Online phase: $\mathcal{R}$ answers $\mathcal{A}$'s oracle queries just as in preprocessing.
- Output: As soon as $\mathcal{A}$ makes its j'-th H query (suppose it is $H(\mathbf{Z})$), $\mathcal{R}$ outputs $\mathbf{Z}$.

Clearly, $\mathcal{R}$ runs in time at most $t_p + T + \theta(\kappa)$ in the preprocessing phase, and time at most t_o in the online phase. Up to the point that $\mathcal{R}$ outputs, $\mathcal{R}$ simulates the CPA game perfectly. If Query happens and $\mathcal{R}$'s guess j' is correct, then $\mathcal{R}$ outputs $\mathbf{Z} = (\mathbf{R}^*)^{2^T}$, solving the T-RSW problem. It follows that

$$\frac{1}{Q_H} \cdot \Pr[\mathsf{Query}] \leq \Pr[\mathcal{R} \text{ wins}] \leq \epsilon_{RSW},$$

hence

$$\Pr[\mathsf{Query}] \leq Q_H \cdot \epsilon_{RSW}.$$

If neither Repeat nor Query happens, then $\mathcal{A}$ does not query $H\left((g^*)^{2^T}\right)$, and (g^*, c^*) is independently random of everything else in $\mathcal{A}$'s view, so b is independent of $\mathcal{A}$'s view. We have that

$$\Pr\left[\mathbf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{TRPKE}} = 1 | \overline{\mathsf{Repeat} \vee \mathsf{Query}}\right] = \frac{1}{2}.$$

Combining the three results above, we get

$$\Pr\left[\mathbf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{TRPKE}} = 1\right] \leq \frac{1}{2} + Q_H \cdot \epsilon_{RSW} + \frac{Q_E}{2^{\theta(\kappa)}},$$

which completes the proof.

D.2 An CCA-secure Construction

Obviously, the TRFE scheme in Section D.1 is malleable (i.e., if the last bit of the ciphertext is flipped, then so is the last bit of the message), and thus not CCA-secure. In this section we present an CCA-secure scheme. At first glance, simply replacing the one-time pad with a CCA-secure secret-key encryption scheme would work, i.e., we let the ciphertext be

$$\left(\mathbf{R}, \mathsf{Enc}'\left(H\left(\mathbf{R}^{2^T}\right), m\right)\right)$$

(where Enc' is the encryption algorithm in the underlying secret-key encryption scheme, and $\mathbf{R} \leftarrow \mathbb{QR}_N$). However, closer scrutiny shows that proving the CCA security of this scheme seems impossible, as the decryption oracle cannot be simulated in a fast way. Concretely, consider an adversary $\mathcal{A}$ who chooses a random m and a random $\mathbf{R} \leftarrow \mathbb{QR}_N$, computes $\mathbf{R}^{2^T}$, and queries $H\left(\mathbf{R}^{2^T}\right)$ in the preprocessing phase; after that $\mathcal{A}$ makes the challenge query, and then inputs $\left(\mathbf{R}, \mathsf{Enc}'\left(H\left(\mathbf{R}^{2^T}\right), m\right)\right)$ to the decryption oracle. The reduction to the T-RSW problem must answer with m. Since the only way to verify whether $\mathcal{A}$ has queried $H\left(\mathbf{R}^{2^T}\right)$ or not is to compute $\mathbf{R}^{2^T}$, the fastest way to answer the decryption oracle query is to compute $\mathbf{R}^{2^T}$ and then run the slow decryption algorithm, which takes time longer than allowed.

Trapdoor VDF. To resolve this issue, we make use of *verifiable delay functions (VDFs)*. A VDF evaluates a function F with needs a prescribed time (e.g., T) to compute, while allowing for much faster verification given the function input and output, as well as an additional proof. We let the underlying function be $F(\mathbf{X}) = \mathbf{X}^{2^T}$, and include the proof π in the random oracle. In this way, the decryption oracle can be simulated in a fast manner: to check if the adversary's RO query is $\mathbf{R}^{2^T}$, the reduction only needs to verify π, which is much faster than computing $\mathbf{R}^{2^T}$ on its own.

In order to enable fast encryption, we need a VDF with a *trapdoor*, with which both the function output and the proof can be computed in a fast manner. The formal definition of a trapdoor VDF is presented below.

Definition 23 (Trapdoor Verifiable Delay Function). *A* $(t_{feval}, t_{seval}, t_{vrfy})$-trapdoor verifiable delay function (trapdoor VDF) *for function* $F : \{0,1\}^* \times \mathcal{X} \to \mathcal{Y}$ *is a tuple of algorithms* $\mathsf{TVDF} = (\mathsf{PGen}, \mathsf{Eval}_f, \mathsf{Eval}_s, \mathsf{Vrfy})$ *with the following behavior:*

- *The randomized* parameter generation *algorithm* PGen *takes as input the security parameter* 1^κ *and outputs public parameters par and a trapdoor td.*
- *The deterministic* fast evaluation algorithm Eval_f *takes as input public parameters par, a trapdoor td, and a function input* $x \in \mathcal{X}$*. It outputs a function output* $y \in \mathcal{Y}$ *and a proof* π *and runs in time at most* t_{feval} *for all* $(par, td) \in \{\mathsf{PGen}(1^\kappa)\}$ *and* $x \in \mathcal{X}$*.*
- *The deterministic* slow evaluation algorithm Eval_s *takes as input public parameters par and a function input* $x \in \mathcal{X}$*. It outputs a function output* $y \in \mathcal{Y}$ *and a proof* π *and runs in time at most* t_{seval} *for all* $(par, \star) \in \{\mathsf{PGen}(1^\kappa)\}$ *and* $x \in \mathcal{X}$*.*
- *The deterministic* verifier algorithm Vrfy *takes as input public parameters par, a function input* $x \in \mathcal{X}$*, a function output* $y \in \mathcal{Y}$*, and a proof* π*. It outputs* 1 *(accept) or* 0 *(reject) and runs in time at most* t_{vrfy} *for all* $x \in \mathcal{X}$*,* $y \in \mathcal{Y}$ *and* π*.*

We require perfect uniqueness*: For all* $(par, td) \in \{\mathsf{PGen}(1^\kappa)\}$ *and all* $x \in \mathcal{X}$*, there exists a proof* π *s.t.* $\mathsf{Eval}_f(par, td, x) = \mathsf{Eval}_s(par, x) = (F_{par}(x), \pi)$ *and* $\mathsf{Vrfy}(par, x, F_{par}(x), \pi) = 1$*.*

Soundness requires that it is hard to generate a "fake" pair of (y, π) for some function input x.

Definition 24 (Soundness). *Let* $\mathsf{TVDF} = (\mathsf{PGen}, \mathsf{Eval}_f, \mathsf{Eval}_s, \mathsf{Vrfy})$ *be a trapdoor VDF for function* F*. The* soundness of TVDF *is defined via the following game* $\mathbf{SND}_{\mathsf{TVDF}}$*:*

- ***Setup:*** $\mathbf{SND}_{\mathsf{TVDF}}$ *samples* (par, td) *via* $(par, td) \leftarrow \mathsf{PGen}(1^\kappa)$*. It runs* $\mathcal{A}$ *on input* par*.*
- ***Output Determination:*** *When* $\mathcal{A}$ *returns* (x, y, π)*,* $\mathbf{SND}_{\mathsf{TVDF}}$ *outputs* 1 *if* $\mathsf{Vrfy}(par, x, y, \pi) = 1$ *and* $(y, \pi) \neq \mathsf{Eval}_s(par, x)$*. It outputs* 0 *otherwise.*

We say that TVDF *is* (ϵ, t)-sound *if for all adversaries* $\mathcal{A}$ *running in time* t*,*

$$\Pr[\mathbf{SND}^{\mathcal{A}}_{\mathsf{TVDF}} = 1] \leq \epsilon.$$ [7]

Pietrzak's VDF. The trapdoor VDF used in our application can be instantiated using Pietrzak's VDF [25], which evaluates the function $F_M(\mathbf{X}) = \mathbf{X}^{2^T}$ (where $\mathbf{X}$ is an element in $\mathbb{QR}_N$). We recall the scheme in Fig. 4, adjusted to our syntax; in particular, we add the fast evaluation algorithm, where the trapdoor is defined as $\phi(N)$.[8]

[25] shows that the scheme in Fig. 4 is $(\frac{3Q}{2^\kappa}, Q)$-sound, where the adversary queries the random oracle H at most Q times.[9]

We are now ready to present our CCA-secure TRFE scheme in Fig. 5.

Theorem 7. *Suppose that*

- *The* T*-RSW problem is* $(\epsilon_{RSW}, t_p, t_o)$*-hard relative to* GenMod*,*
- TVDF *is an* $(\epsilon_{SND}, t_p + t_o)$*-sound trapdoor VDF, and*
- AE *is an* $(\epsilon_{CPA}, t_p, t_o)$*-CPA-secure and* $(\epsilon_{CTXT}, t_p + t_o)$*-INT-CTXT secure authenticated encryption scheme.*

Then the TRFE scheme in Fig. 5 is $(\epsilon_{SND} + Q_D \cdot \epsilon_{CTXT} + \epsilon_{CPA} + \epsilon_{RSW}, t_p, t_o)$*-CCA-secure, where the adversary queries the decryption oracle* DEC *at most* Q_D *times.*

Proof. Let $\mathcal{A}$ be an adversary in the CCA Security Game for this TLE scheme, $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$, with preprocessing time t_p and online time t_o. Suppose $\mathcal{A}$'s challenge ciphertext is $(\mathbf{R}^*, c^*)$. The proof goes by a sequence of games, which we describe next.

$\mathcal{G}_0$: $\mathcal{G}_0$ is the original CCA Security Game, $\mathbf{IND\text{-}CCA}_{\mathsf{TRPKE}}$.

$\mathcal{G}_1$: $\mathcal{G}_1$ is identical to $\mathcal{G}_0$, except that (1) when $\mathcal{A}$ makes an $H(\mathbf{R}, \mathbf{Z}, \pi)$ query, check if $\mathsf{Vrfy}(N, \mathbf{R}, \mathbf{Z}, \pi) = 1$, and if so (call such query a "crucial query for $\mathbf{R}$"), record $(\mathbf{R}, \mathbf{Z}, \pi)$; (2) abort if there is more than one "crucial query" for the some $\mathbf{R}$.

Let Fake be the event that $\mathcal{A}$ makes an $H(\mathbf{R}, \mathbf{Z}, \pi)$ query s.t. $(\mathbf{Z}, \pi) \neq \mathsf{Eval}_s(N, \mathbf{R})$ but $\mathsf{Vrfy}(\mathbf{R}, \mathbf{Z}, \pi) = 1$. We can see that $\mathcal{G}_1$ and $\mathcal{G}_0$ are identical unless (a subevent of) Fake happens. We upper bound $\Pr[\mathsf{Fake}]$ by constructing a reduction $\mathcal{R}_{SND}$ to the soundness property of TVDF:

[7] The standard notion of soundness [4] only requires that y cannot be "faked", i.e., in $\mathbf{SND}_{\mathsf{TVDF}}$ condition $(y, \pi) \neq \mathsf{Eval}_s(par, x)$ is replaced by $y \neq F_{par}(x)$. However, in our application we need the property that π cannot be "faked" either.

[8] The scheme is defined for arbitrary $T \in \mathbb{Z}^+$, but in the presentation we assume that T is a power of 2 for simplicity.

[9] In fact, [25] only shows standard soundness, but our stronger soundness property can be derived via inspection of the proof.

Let $H : \mathbb{QR}_N \times \mathbb{QR}_N \times \mathbb{Z}^+ \times \mathbb{QR}_N \to \mathsf{Primes}(\kappa)$ be a hash function (modeled as a random oracle), where $\mathsf{Primes}(\kappa)$ is the set of the first κ prime numbers.

- $\mathsf{PGen}(1^\kappa)$: Run $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$, compute $\phi := \phi(N) = (p-1)(q-1)$ and output $par := (N, \phi)$.
- $\mathsf{Eval}_s(N, \mathbf{X})$: Compute $\mathbf{Y} := \mathsf{RepSqr}(N, \mathbf{X}, 2^T \bmod \frac{\phi}{4})$. For $i = 1, \ldots, \log T$, compute

$$\mathbf{V}_i := \mathsf{RepSqr}(N, \mathbf{X}_{i-1}, 2^{T/2^i} \bmod \frac{\phi}{4}), r_i := H\left(\mathbf{X}, \mathbf{Y}, \frac{T}{2^i}, \mathbf{V}_i\right),$$

$$\mathbf{X}_i := \mathsf{RepSqr}(N, \mathbf{X}_{i-1}, r_i \bmod \frac{\phi}{4}) \cdot \mathbf{V}_i, \mathbf{Y}_i := \mathsf{RepSqr}(N, \mathbf{V}_i, r_i \bmod \frac{\phi}{4}) \cdot \mathbf{Y}_{i-1}$$

  (where $\mathbf{X}_0 = \mathbf{X}$), and output $(\mathbf{V}_1, \ldots, \mathbf{V}_{\log T})$.
- $\mathsf{Eval}_s(N, \mathbf{X})$: Compute $\mathbf{Y} := \mathsf{RepSqr}(N, \mathbf{X}, 2^T)$. For $i = 1, \ldots, \log T$, compute

$$\mathbf{V}_i := \mathsf{RepSqr}(N, \mathbf{X}_{i-1}, 2^{T/2^i}), r_i := H\left(\mathbf{X}, \mathbf{Y}, \frac{T}{2^i}, \mathbf{V}_i\right),$$

$$\mathbf{X}_i := \mathsf{RepSqr}(N, \mathbf{X}_{i-1}, r_i) \cdot \mathbf{V}_i, \mathbf{Y}_i := \mathsf{RepSqr}(N, \mathbf{V}_i, r_i) \cdot \mathbf{Y}_{i-1}$$

  (where $\mathbf{X}_0 = \mathbf{X}$), and output $(\mathbf{V}_1, \ldots, \mathbf{V}_{\log T})$.
- $\mathsf{Vrfy}(N, \mathbf{X}, \mathbf{Y}, (\mathbf{V}_1, \ldots, \mathbf{V}_{\log T}))$: For $i = 1, \ldots, \log T$, compute

$$r_i := H\left(\mathbf{X}, \mathbf{Y}, \frac{T}{2^i}, \mathbf{V}_i\right),$$

$$\mathbf{X}_i := \mathsf{RepSqr}(N, \mathbf{X}_{i-1}, r_i) \cdot \mathbf{V}_i, \mathbf{Y}_i := \mathsf{RepSqr}(N, \mathbf{V}_i, r_i) \cdot \mathbf{Y}_{i-1}$$

  (where $\mathbf{X}_0 = \mathbf{X}$). If $\mathbf{Y}_{\log T} = \mathbf{X}_{\log T}^2$, then output 1, otherwise output 0.

Fig. 4. Pietrzak's $(\theta(\kappa) + 2\sqrt{\theta(\kappa)}, T + 2\sqrt{T}, 2\log T)$-trapdoor VDF

Let ℓ be polynomial in κ, and $H : \mathbb{QR}_N \to \{0,1\}^\ell$ be a hash function (modeled as a random oracle). Let $\mathsf{TVDF} = (\mathsf{PGenVDF}, \mathsf{Eval}_f, \mathsf{Eval}_s, \mathsf{Vrfy})$ be a $(t_{feval}, t_{seval}, t_{vrfy})$-trapdoor VDF for the function $F_N(\mathbf{X}) = \mathbf{X}^{2^T}$ where $\mathsf{PGenVDF}$ is the same with PGen below, and $\mathsf{AE} = (\mathsf{PGen}', \mathsf{KGen}', \mathsf{Enc}', \mathsf{Dec}')$ be a (t'_e, t'_d)-authenticated encryption scheme where $\mathsf{PGen}'(1^\kappa)$ outputs ℓ and $\mathsf{KGen}'(\ell)$ outputs a random string in $\{0,1\}^\ell$.

- $\mathsf{PGen}(1^\kappa)$: Run $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$, compute $\phi := \phi(N) = (p-1)(q-1)$ and output $par := (N, \phi)$.
- $\mathsf{KGen}(N, \phi)$: Output $(sk := (N, \phi), pk := N)$.
- $\mathsf{Enc}_f((N, \phi), m)$: Choose $\mathbf{R} \leftarrow \mathbb{QR}_N$, compute $(\mathbf{Z}, \pi) := \mathsf{Eval}_f(N, \phi, \mathbf{R})$, and output $(\mathbf{R}, c \leftarrow \mathsf{Enc}'(H(\mathbf{R}, \mathbf{Z}, \pi), m))$.
- $\mathsf{Enc}_s(N, m)$: Choose $\mathbf{R} \leftarrow \mathbb{QR}_N$, compute $(\mathbf{Z}, \pi) := \mathsf{Eval}_s(N, \mathbf{R})$, and output $(\mathbf{R}, c \leftarrow \mathsf{Enc}'(H(\mathbf{R}, \mathbf{Z}, \pi), m))$.
- $\mathsf{Dec}_f((N, \phi), (\mathbf{R}, c))$: Compute $(\mathbf{Z}, \pi) := \mathsf{Eval}_f(N, \phi, \mathbf{R})$ and output $\mathsf{Dec}'(H(\mathbf{R}, \mathbf{Z}, \pi), c)$.
- $\mathsf{Dec}_s((N, \phi), (\mathbf{R}, c))$: Compute $(\mathbf{Z}, \pi) := \mathsf{Eval}_s(N, \mathbf{R})$ and output $\mathsf{Dec}'(H(\mathbf{R}, \mathbf{Z}, \pi), c)$.

Fig. 5. An CCA-secure $(t_{feval} + t'_e + \theta(\kappa), t_{seval} + t'_e + \theta(\kappa), t_{feval} + t_d, t_{seval} + t_d)$-TLE scheme

$\mathcal{R}_{SND}$ runs the code of $\mathcal{G}_1$ (including PGen, so $\mathcal{R}_{SND}$ knows ϕ and hence can answer $\mathcal{A}$'s ENC queries in a fast way), except that when $\mathcal{A}$ queries $H(\mathbf{R}, \mathbf{Z}, \pi)$, $\mathcal{R}_{SND}$ checks if Fake happens, and if so, it outputs $(\mathbf{R}, \mathbf{Z}, \pi)$ (and halts).

Clearly, $\mathcal{R}_{SND}$ runs in time at most $t_p + t_o$. If Fake happens, then up to the point that $\mathcal{R}_{SND}$ outputs, $\mathcal{R}_{SND}$ simulates the CCA game perfectly. Furthermore, $\mathcal{R}_{SND}$ outputs a tuple s.t. $(\mathbf{Z}, \pi) \neq \mathsf{Eval}_s(N, \mathbf{R})$ but $\mathsf{Vrfy}(\mathbf{R}, \mathbf{Z}, \pi) = 1$, breaking the soundness property of TVDF. It follows that

$$|\Pr[\mathcal{G}_1^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_0^{\mathcal{A}} = 1]| \leq \Pr[\mathsf{Fake}] \leq \Pr[\mathcal{R}_{SND} \text{ wins}] \leq \epsilon_{SND}.$$

$\mathcal{G}_2$: $\mathcal{G}_2$ is identical to $\mathcal{G}_1$, except that when $\mathcal{A}$ makes a "new" decryption oracle query $\mathsf{DEC}(\mathbf{R}, c)$ (i.e., $(\mathbf{R}, c)$ is not the answer of any of $\mathcal{A}$'s previous encryption oracle ENC queries), check if there is a record $(\mathbf{R}, \mathbf{Z}, \pi)$ (i.e., $\mathcal{A}$ has previously made a "crucial query" for $\mathbf{R}$). If not, then output $\perp$ to $\mathcal{A}$; otherwise output $\mathsf{Dec}'(H(\mathbf{R}, \mathbf{Z}, \pi), c)$ to $\mathcal{A}$.

For every $\mathcal{A}$'s "new" DEC query $\mathsf{DEC}(\mathbf{R}, c)$, let $(\mathbf{Z}, \pi) := \mathsf{Eval}_s(N, \mathbf{R}) = \mathsf{Eval}_f(N, \phi, \mathbf{R})$. There are two possible cases:

- $\mathcal{A}$ has previously made a "crucial query" $H(\mathbf{R}, \mathbf{Z}, \pi)$: Then both $\mathcal{G}_2$ and $\mathcal{G}_1$ output $\mathsf{Dec}'(H(\mathbf{R}, \mathbf{Z}, \pi), c)$, and thus are identical.
- $\mathcal{A}$ has not made a "crucial query" for $\mathbf{R}$: Then $\mathcal{G}_2$ outputs $\perp$, while $\mathcal{G}_1$ outputs $\mathsf{Dec}'\left(H\left(\mathbf{R}, \mathbf{Z}, \pi\right), c\right)$.

(Note that $\mathcal{A}$ could not have made more than one "crucial query" for $\mathbf{R}$; otherwise both $\mathcal{G}_2$ and $\mathcal{G}_1$ would have aborted earlier.)

We can see that $\mathcal{G}_2$ and $\mathcal{G}_1$ are identical unless there exists an $\mathcal{A}$'s "new" DEC query $\mathsf{DEC}(\mathbf{R}, c)$ s.t. $\mathcal{A}$ has not previously made a "crucial query" for $\mathbf{R}$, but $\mathsf{Dec}'(H(\mathbf{R}, \mathbf{Z}, \pi), c) \neq \perp$. Denote such event $\mathsf{NotAbort}$. Since $\mathcal{A}$ does not query $H(\mathbf{R}, \mathbf{Z}, \pi)$, its output is random in $\mathcal{A}$'s view; by the INT-CTXT property of AE, it is hard for $\mathcal{A}$ to come up with a valid ciphertext c under $H(\mathbf{R}, \mathbf{Z}, \pi)$. Formally, we upper-bound $\Pr[\mathsf{NotAbort}]$ by constructing a reduction $\mathcal{R}_{CTXT}$ to the INT-CTXT property of AE:

$\mathcal{R}_{CTXT}$ chooses $i' \leftarrow \{1, \ldots, Q_D\}$ (a guess that $\mathsf{NotAbort}$ happens for the first time when $\mathcal{A}$ makes its i'-th "new" DEC query). Then $\mathcal{R}_{CTXT}$ runs the code of $\mathcal{G}_2$ (including PGen, so $\mathcal{R}_{CTXT}$ knows ϕ and hence can answer $\mathcal{A}$'s ENC queries in a fast way), except that in $\mathcal{A}$'s i'-th "new" DEC query $\mathsf{DEC}(\mathbf{R}, c)$, $\mathcal{R}_{CTXT}$ checks if $\mathcal{A}$ has not made a "crucial query" for $\mathbf{R}$, and if so, $\mathcal{R}_{CTXT}$ outputs c (otherwise $\mathcal{R}_{CTXT}$ aborts).

Clearly, $\mathcal{R}_{CTXT}$ runs in time at most $t_p + t_o$. If $\mathsf{NotAbort}$ happens and $\mathcal{R}_{CTXT}$'s guess i' is correct, then up to the point that $\mathcal{R}_{CTXT}$ outputs, $\mathcal{R}_{CTXT}$ simulates the CCA game perfectly. Furthermore, $\mathcal{R}_{CTXT}$ outputs a valid ciphertext c under $H(\mathbf{R}, \mathbf{Z}, \pi)$ (which is a random string in $\{0,1\}^\ell$ in $\mathcal{R}_{CTXT}$'s view), breaking the INT-CTXT property of AE. It follows that

$$\frac{1}{Q_D} \cdot \Pr[\mathsf{NotAbort}] \leq \Pr[\mathcal{R}_{AUTH} \text{ wins}] \leq \epsilon_{CTXT},$$

hence

$$|\Pr[\mathcal{G}_2^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_1^{\mathcal{A}} = 1]| \leq \Pr[\mathsf{NotAbort}] \leq Q_D \cdot \epsilon_{CTXT}.$$

$\mathcal{G}_2$ makes $\mathcal{A}$'s decryption oracle DEC essentially useless, because the only case that DEC does not return $\perp$ is that $\mathcal{A}$ has made a "crucial query", but then $\mathcal{A}$ could decrypt on its own.

$\mathcal{G}_3$: $\mathcal{G}_3$ is identical to $\mathcal{G}_2$, except that (1) when $\mathcal{A}$ makes an encryption oracle query $\mathsf{ENC}(m)$, choose $\mathbf{R} \leftarrow \mathbb{QR}_N$ and $h \leftarrow \{0,1\}^\ell$, record $(\mathbf{R}, h)$, compute $c \leftarrow \mathsf{Enc}'(h, m)$, and output $(\mathbf{R}, c)$ to $\mathcal{A}$; and (2) when $\mathcal{A}$ makes an H query $H(\mathbf{R}, \mathbf{Z}, \pi)$, check if $\mathsf{Vrfy}(\mathbf{R}, \mathbf{Z}, \pi) = 1$ (i.e., if it is a "crucial query" for $\mathbf{R}$), and if there is a record $(\mathbf{R}, h)$. If both checks pass, then "program" $H(\mathbf{R}, \mathbf{Z}, \pi) := h$ (otherwise this H query is answered via lazy sampling, just as in $\mathcal{G}_2$).

The only difference between $\mathcal{G}_3$ and $\mathcal{G}_2$ is that $\mathcal{G}_2$ answers $\mathcal{A}$'s $\mathsf{ENC}(m)$ with $(\mathbf{R}, c)$ where $c \leftarrow \mathsf{Enc}'(H(\mathbf{R}, \mathbf{Z}, \pi), m)$, while in $\mathcal{G}_3$ $c \leftarrow \mathsf{Enc}'(h, m)$ where $h \leftarrow \{0,1\}^\ell$, and $H(\mathbf{R}, \mathbf{Z}, \pi)$ is "programmed" to be h if $\mathcal{A}$ queries it. This is merely a conceptual change, so

$$\Pr[\mathcal{G}_3^{\mathcal{A}} = 1] = \Pr[\mathcal{G}_2^{\mathcal{A}} = 1].$$

$\mathcal{G}_4$: Recall that $\mathcal{A}$'s challenge ciphertext is $(\mathbf{R}^*, c^*)$, and let $(\mathbf{Z}^*, \pi^*) := \mathsf{Eval}_s(N, \mathbf{R}^*) = \mathsf{Eval}_f(N, \phi, \mathbf{R}^*)$ (so $\mathbf{Z}^* = (\mathbf{R}^*)^{2^T}$). $\mathcal{G}_4$ is identical to $\mathcal{G}_3$, except that $\mathcal{G}_4$ aborts if $\mathcal{A}$ queries $H(\mathbf{R}^*, \mathbf{Z}, \pi^*)$. Denote such event Query.

We upper bound $\Pr[\mathsf{Query}]$ by constructing a reduction $\mathcal{R}_{RSW}$ to the T-RSW problem (where the inputs are N and $\mathbf{R}^*$):

- Preprocessing phase: $\mathcal{R}_{RSW}$, on input N, runs $\mathcal{A}(N)$ and answers $\mathcal{A}$'s oracle queries as in $\mathcal{G}_3$.
- Challenge query: When $\mathcal{A}$ makes its challenge query on (m_0, m_1), $\mathcal{R}_{RSW}$ asks for $\mathbf{R}^*$ from the RSW challenger, chooses $h^* \leftarrow \{0,1\}^\ell$, computes $c^* \leftarrow \mathsf{Enc}'(h^*, m_b)$ and returns $(\mathbf{R}^*, c^*)$.
- Online phase: $\mathcal{R}_{RSW}$ answers $\mathcal{A}$'s encryption queries, H queries and DEC queries as in $\mathcal{G}_3$.
- Output: When $\mathcal{A}$ makes a query $H(\mathbf{R}^*, \mathbf{Z}, \pi)$, $\mathcal{R}_{RSW}$ checks if $\mathsf{Vrfy}(\mathbf{R}^*, \mathbf{Z}, \pi) = 1$, and if so, it outputs $\mathbf{Z}$.

Clearly, $\mathcal{R}_{RSW}$ runs in time at most t_p in the preprocessing phase, and time at most t_o in the online phase. $\mathcal{R}_{RSW}$ simulates $\mathcal{G}_3$ perfectly, and if Query happens, $\mathcal{R}_{RSW}$ outputs $\mathbf{Z} = (\mathbf{R}^*)^{2^T}$, solving the RSW problem. It follows that

$$|\Pr[\mathcal{G}_4^{\mathcal{A}} = 1] - \Pr[\mathcal{G}_3^{\mathcal{A}} = 1]| \leq \Pr[\mathsf{Query}] \leq \Pr[\mathcal{R}_{RSW} \text{ wins}] \leq \epsilon_{RSW}.$$

Now we analyze $\mathcal{A}$'s advantage in $\mathcal{G}_4$. Since $\mathcal{A}$ is not allowed to query $H(\mathbf{R}^*, \mathbf{Z}, \pi^*)$ (denote the output h^*), h^* is random in $\mathcal{A}$'s view; by the CPA property of AE, $\mathsf{Enc}'(h^*, m_0)$ and $\mathsf{Enc}'(h^*, m_1)$ are indistinguishable. Formally, we upper bound $\mathcal{A}$'s advantage by constructing a reduction $\mathcal{R}_{CPA}$ to the CPA property of AE:

$\mathcal{R}_{CPA}$ runs the code of $\mathcal{G}_4$, except that when $\mathcal{A}$ makes its challenge query on (m_0, m_1), $\mathcal{R}_{CPA}$ also makes its challenge query on (m_0, m_1), and on its challenge ciphertext c^*, $\mathcal{R}_3$ chooses $\mathbf{R}^* \leftarrow \mathbb{QR}_N$ and sends $(\mathbf{R}^*, c^*)$ to $\mathcal{A}$. $\mathcal{R}_{CPA}$ copies $\mathcal{A}$'s output bit b'.

Clearly, $\mathcal{R}_{CPA}$ runs in time at most t_p in the preprocessing phase, and time at most t_o in the online phase. $\mathcal{R}_{CPA}$ simulates $\mathcal{G}_4$ perfectly, and if $\mathcal{A}$ wins, then $\mathcal{R}_{CPA}$ wins. It follows that

$$\Pr[\mathcal{G}_4^{\mathcal{A}} = 1] = \Pr[\mathcal{R}_{CPA} \text{ wins}] \leq \frac{1}{2} + \epsilon_{CPA}.$$

Summing up all results above, we conclude that

$$\Pr\left[\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TRPKE}} = 1\right] \leq \frac{1}{2} + \epsilon_{SND} + Q_D \cdot \epsilon_{CTXT} + \epsilon_{CPA} + \epsilon_{RSW},$$

which completes the proof.

E Deferred Definitions for NITICs

In this section, we give definitions for binding and soundness properties of a NITIC. Similarly as for our definition of hiding, we also define them relative to a decommit oracle that allows the adversary to open arbitrary commitments.

Binding. The binding property states that the verifier cannot be convinced that $C \leftarrow \mathsf{Com}(crs, m)$ is a commitment of some other message $m' \neq m$.

$\underline{\mathbf{BND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}}}$:

- **Setup: BND-CCA$_{\mathsf{TC}}$** samples a common reference string crs via $crs \leftarrow \mathsf{PGen}(1^\kappa)$.
- **Online Phase: BND-CCA$_{\mathsf{TC}}$** runs $\mathcal{A}$ on input crs. In this phase, $\mathcal{A}$ is given access to a *decommit oracle* DEC, which on input C, returns $\mathsf{FDecom}(crs, C)$ in time t_{dc}.
- **Output Determination:** When $\mathcal{A}$ outputs (m, r, C, m', π'), **BND-CCA$_{\mathsf{TC}}$** returns 1 if $m' \neq m$, $C = \mathsf{Com}(crs, m; r)$, and $\mathsf{DecomVrfy}(crs, C, m', \pi') = 1$. It returns 0 otherwise.

Definition 25 (BND-CCA Security for Commitments). *We say that a NITIC scheme* TC *is* (ϵ, t, t_{dc})-BND-CCA-secure *if for all adversaries* $\mathcal{A}$ *running in time* t*,*

$$\Pr\left[\mathbf{BND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}} = 1\right] \leq \epsilon.$$

It is easy to see that our NITIC scheme in Fig. 2 actually has *perfect* binding, i.e., there does not exist (m, r, C, m', π') s.t. $m' \neq m$, $C = \mathsf{Com}(crs, m; r)$, and $\mathsf{DecomVrfy}(crs, C, m', \pi') = 1$. Indeed, $C = \mathsf{Com}(crs, m; r)$ implies that $m = \mathsf{Dec}_s(pk, C)$, and $\mathsf{DecomVrfy}(crs, C, m', \pi') = 1$ implies $m' = \mathsf{Dec}_s(pk, C)$, so if both conditions hold, then it must be the case that $m = m'$.

Soundness. For soundness, we essentially wish to capture the requirement that the adversary cannot produce a valid commitment (no matter with how much time it is given) which cannot be forcibly decommitted.

$\underline{\mathbf{SND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}}}$:

- **Setup: SND-CCA$_{\mathsf{TC}}$** samples a common reference string crs via $crs \leftarrow \mathsf{PGen}(1^\kappa)$.
- **Online Phase: SND-CCA$_{\mathsf{TC}}$** runs $\mathcal{A}$ on input crs. In this phase, $\mathcal{A}$ is given access to a *decommit oracle* DEC, which on input C, returns $\mathsf{FDecom}(crs, C)$ in time t_{dc}.
- **Output Determination:** When $\mathcal{A}$ outputs (C, π), **SND-CCA$_{\mathsf{TC}}$** returns 1 if $\mathsf{ComVrfy}(crs, C, \pi) = 1$ and $\mathsf{FDecom}(crs, C) = \perp$. It returns 0 otherwise.

Definition 26 (SND-CCA Security for Commitments). *We say that a NITIC scheme* TC *is* (ϵ, t)-SND-CCA-secure *if for all adversaries* $\mathcal{A}$ *running in time* t *in* **SND-CCA$_{\mathsf{TC}}$***,*

$$\Pr\left[\mathbf{SND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}} = 1\right] \leq \epsilon.$$

In our NITIC scheme in Fig. 2, $\mathsf{FDecom}(crs, C) = \perp$ implies that $\mathsf{Dec}_s(pk, C) = \perp$, i.e., there does not exist (m, r) s.t. $C = \mathsf{Enc}(pk, m; r)$. On the other hand, $\mathsf{ComVrfy}(crs, C, \pi)$ outputs $\mathsf{Vrfy}_{\mathsf{Com}}(crs_{\mathsf{Com}}, c, \pi_{\mathsf{Com}})$, which is the verification algorithm for the relation $R_{\mathsf{Com}} = \{(c, (m, r)) | c = \mathsf{Enc}(pk, m; r)\}$. Therefore, soundness of our NITIC scheme is directly implied by soundness of the underlying $\mathsf{NIZK}_{\mathsf{Com}}$ scheme.