On the Security of Time-Lock Puzzles and Timed Commitments

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

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

Abstract. Time-lock puzzles—problems whose solution requires some amount of *sequential* effort—have recently received increased interest (e.g., in the context of verifiable delay functions). Most constructions rely on the sequential-squaring conjecture that computing $g^{2^T} \bmod N$ for a uniform g requires at least T (sequential) steps. We study the security of time-lock primitives from two perspectives:

1. We give the first hardness result about the sequential-squaring conjecture in a non-generic model. Namely, in a quantitative version of the algebraic group model (AGM) that we call the *strong* AGM, we show that speeding up 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-lock puzzles. We extend existing security definitions to settings that may arise when using timed commitments in higher-level protocols, and give the first construction of *non-malleable* timed commitments. As a building block of independent interest, we also define (and give constructions for) a related primitive called *timed public-key encryption*.

1 Introduction

Time-lock puzzles, introduced by Rivest, Shamir, and Wagner [29], refer to a fascinating type of computational problem that requires a certain amount of sequential effort to solve. Time-lock puzzles can be used to construct timed commitments [7], which "send a message m into the future" in the sense that m remains computationally hidden for some time T, but can be recovered once time T has passed. Time-lock puzzles can be used to build various other primitives, including verifiable delay functions (VDFs) [5,6,28,33], zero-knowledge proofs [14], and non-malleable (standard) commitments [19], and have applications to fair coin tossing, e-voting, auctions, and contract signing [7, 23]. In this work, we (1) provide formal evidence, in a non-generic model, in support of the hardness of the most widely used time-lock puzzle and (2) give new, stronger security definitions (and constructions) for timed commitments and related primitives. These contributions are explained in more detail next.

⋆ Portions of this work were done while at George Mason University.

Hardness in the (strong) AGM. The hardness assumption underlying the most popular time-lock puzzle [29] is that, given a uniform generator g in the group of quadratic residues[1] $\mathbb{QR}_N$ (where N is the product of two safe primes), it is hard to compute $g^{2^T} \bmod N$ in fewer than T sequential steps. We study this assumption in a new, strengthened version of the algebraic group model (AGM) [16] that we call the *strong AGM (SAGM)*, which lies between the generic group model (GGM) [24, 32] and the AGM. Roughly, an algorithm $\mathcal{A}$ in the AGM is allowed to utilize the *actual bitstrings* representing group elements in the course of its computation (something that is not allowed in the GGM), but is constrained in the sense that for any group element h that $\mathcal{A}$ outputs, $\mathcal{A}$ must also output coefficients showing how h was computed from group elements previously given to $\mathcal{A}$ as input. The SAGM imposes the stronger constraint that $\mathcal{A}$ output the *entire path of its computation* (i.e., all intermediate group operations) that resulted in h. We show that for strongly algebraic algorithms, computing $g^{2^T} \bmod N$ from g using fewer than T group operations is as hard as factoring N. We also show that it is not possible to reduce the hardness of sequential squaring to factoring in the AGM (assuming factoring is hard in the first place). Our result is the first formal argument for the sequential hardness of squaring in a non-generic model, and immediately implies the security of Pietrzak's VDF [28] in the SAGM (assuming the hardness of factoring).

Rotem and Segev [30] recently analyzed the hardness of sequential squaring and related functions over $\mathbb{Z}_N$ in the *generic ring model* [1], where an algorithm can perform generic ring additions and multiplications but does not get access to the actual bitstrings representing ring elements. This makes their analysis incomparable to our analysis in the strong AGM.

Non-malleable timed commitments. The second part of our paper is concerned with the security of *non-interactive timed commitments* (NITCs). 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 after a certain amount of time in case the committer refuses to open it. Moreover, a commitment comes with a proof that it can be forced open if needed. We introduce a strong notion of non-malleability for such schemes. To construct a non-malleable NITC, we formalize as a stepping stone a timed public-key analogue that we call *timed public-key encryption* (TPKE). We then show how to achieve an appropriate notion of CCA-security for TPKE. Finally, we show a generic transformation from CCA-secure TPKE to non-malleable NITC. Although our main purpose for introducing TPKE is to obtain a non-malleable NITC, we believe that TPKE is an independently interesting primitive worthy of further study.

1.1 Related Work

We highlight here additional work not already cited earlier. Mahmoody et al. [22] show constructions of time-lock puzzles in the random-oracle model, and Bitan-

[1] The problem was originally stated over the ring $\mathbb{Z}_N$. Subsequent works have studied it both over $\mathbb{QR}_N$ [28] and $\mathbb{J}_N$ (elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$) [23].

sky et al. [4] give constructions based on randomized encodings. In recent work, Malavolta and Thyagarajan [23] study a homomorphic variant of time-lock puzzles. Another line of work initiated by May [25] and later formalized by Rivest et al. [29] studies a model for timed message transmission between a sender and receiver in the presence of a trusted server. Bellare and Goldwasser [3] considered a notion of "partial key escrow" in which a server can store keys in escrow and learn only some of them unless it expends significant computational effort; this model was subsequently studied by others [11,13] as well. Liu et al. [21] propose a time-lock encryption scheme based on witness encryption in a model with a global clock.

Concurrent work. In work concurrent with our own, Baum et al. [2] formalize time-lock puzzles and timed commitments in the framework of universal composability (UC) [9]; UC timed commitments are presumably also non-malleable. Baum et al. present constructions in the (programmable) random-oracle model that achieve their definitions, and show that their definitions are impossible to realize in the plain model. Ephraim et al. [15] recently formalized a notion of non-malleable timed commitments that is somewhat different from our own: they do not distinguish between time-lock puzzles and timed commitments, which makes a direct comparison somewhat difficult. They give a generic construction of a time-lock puzzle from a VDF in the random-oracle model.

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 sequential squaring. We give definitions for NTIC and TPKE in Section 4.1, and give a construction of CCA-secure TPKE in Section 4.2. In Section 4.3, we then show a simple, generic conversion from CCA-secure TPKE to non-malleable NITC.

2 Notation and Preliminaries

Notation. We use ":=" to denote a deterministic assignment and "$\leftarrow$" to denote assignment via a randomized process. 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 $||n||$. We denote the security parameter by κ. We write $\mathsf{Expt}^{\mathcal{A}}$ for the output of experiment Expt involving adversary $\mathcal{A}$.

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 the group $\mathbb{QR}_N$ under consideration takes unit time. All algorithms are assumed to have arbitrary parallel computing resources.

2.1 Repeated Squaring and the RSW Problem

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., such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are also prime) with $||p|| = ||q|| = \tau(\kappa)$; here, $\tau(\kappa)$ is defined such that the fastest factoring algorithm takes time 2^κ to factor N with probability $\frac{1}{2}$. GenMod may fail with negligible probability, but we ignore this from now on. It is well known that $\mathbb{QR}_N$ is cyclic with $|\mathbb{QR}_N| = \frac{\phi(N)}{4} = \frac{(p-1)(q-1)}{4}$.

For completeness, we define the factoring problem.

Definition 1. *For an algorithm $\mathcal{A}$, define experiment $\mathbf{FAC}^{\mathcal{A}}_{\mathsf{GenMod}}$ as follows:*

1. *Compute $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$, and then run $\mathcal{A}$ on input N.*
2. *$\mathcal{A}$ outputs $p', q' \notin \{1, N\}$, and the experiment evaluates to 1 iff $N = p'q'$.*

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

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

The repeated squaring algorithm. Given an element $g \in \mathbb{QR}_N$, it is possible to compute $g^1, g^2, g^3, \ldots, g^{2^i}$ (all modulo N) in i steps: in step i, simply multiply each value $g^1, \ldots, g^{2^{i-1}}$ by $g^{2^{i-1}}$. (Recall that we allow unbounded parallelism.) In particular, it is possible to compute g^x for any positive integer x in $\lceil \log x \rceil$ steps. We denote by RepSqr the algorithm that on input (g, N, x) computes g^x in this manner.

Given a generator g of $\mathbb{QR}_N$, it is possible to sample a uniform element of $\mathbb{QR}_N$ 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) is known; if this is not the case, one can instead sample $x \leftarrow \mathbb{Z}_{N^2}$, which results in a negligible statistical difference that we ignore for simplicity. Sampling a uniform element of $\mathbb{QR}_N$ in this way takes at most

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

steps. We denote by $\theta(\kappa) = 4\tau(\kappa)$ the time to sample a uniform element of $\mathbb{QR}_N$.

The RSW problem. We next formally define the repeated squaring problem with preprocessing. This problem was first proposed by Rivest, Shamir, and Wagner [29] and hence we refer to it as the *RSW problem*. We write elements of $\mathbb{G}$ (except for the fixed generator g) using bold, upper-case letters.

Definition 2. *For a stateful algorithm $\mathcal{A}$, define experiment T-$\mathbf{RSW}^{\mathcal{A}}_{\mathsf{GenMod}}$ as follows:*

1. *Compute $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$.*
2. *Run $\mathcal{A}$ on input N in a preprocessing phase to obtain intermediate state.*
3. *Sample $g \leftarrow \mathbb{QR}_N$ and run $\mathcal{A}$ on input g in the online phase.*
4. *$\mathcal{A}$ outputs $\mathbf{X} \in \mathbb{QR}_N$, and the experiment evaluates to 1 iff $\mathbf{X} = g^{2^T} \bmod N$.*

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

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

Clearly, an adversary $\mathcal{A}$ can run $\mathsf{RepSqr}(g, N, 2^T)$ to compute $g^{2^T} \bmod N$ in T steps. This means the T-RSW problem is easy when $t_o \geq T$. In Section 3.1 we show that in the strong algebraic group model, when $t_o < T$ the T-RSW problem is (t_p, t_o, ϵ)-hard (for negligible ϵ) unless N can be factored in time roughly $t_p + t_o$. To put it another way, the fastest stringly algebraic algorithm for computing $g^{2^T} \bmod N$ (short of factoring N) is to run $\mathsf{RepSqr}(g, N, 2^T)$.

We also introduce a *decisional* variant of the RSW assumption where, roughly speaking, the problem is to distinguish $g^{2^T} \bmod N$ from a uniform element of $\mathbb{QR}_N$ in fewer than T steps.

Definition 3. *For a stateful algorithm* $\mathcal{A}$*, define experiment* $T\text{-}\mathbf{DRSW}^{\mathcal{A}}_{\mathsf{GenMod}}$ *as follows:*

1. *Compute* $(N, p, q) \leftarrow \mathsf{GenMod}(1^\kappa)$.
2. *Run* $\mathcal{A}$ *on input* N *in a preprocessing phase to obtain intermediate state.*
3. *Sample* $g, \mathbf{X} \leftarrow \mathbb{QR}_N$ *and a uniform bit* $b \leftarrow \{0, 1\}$*. If* $b = 0$*, run* $\mathcal{A}$ *on inputs* $g, \mathbf{X}$*; if* $b = 1$*, run* $\mathcal{A}$ *on inputs* $g, g^{2^T} \bmod N$ *in the online phase.*
4. $\mathcal{A}$ *outputs a bit* b'*, and the experiment evaluates to* 1 *iff* $b' = b$*.*

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

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

The decisional T-RSW problem is related to the generalized BBS (GBBS) assumption introduced by Boneh and Naor [7]; however, there are several differences. First, the adversary in the GBBS assumption is given the group elements $g, g^2, g^4, g^{16}, g^{256}, \ldots, g^{2^{2^k}}$ and then asked to distinguish $g^{2^{2^{k+1}}}$ from uniform. Second, the GBBS assumption does not account for any preprocessing. Our definition is also similar to the strong sequential squaring assumption [23] except that we do not give g to $\mathcal{A}$ in the preprocessing phase.

2.2 Simulation-Sound NIZK

We recall the notion of simulation-sound non-interactive zero-knowledge (NIZK) proofs [12]. Our definitions are standard except that we explicitly consider the running times of various algorithms involved.

Let $\mathcal{L}_R$ be a NP language defined by relation R. Let $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ be a tuple of algorithms where

– The *parameter-generation algorithm* GenZK takes as input the security parameter 1^κ and outputs a common reference string crs.
– The *prover algorithm* Prove takes as input a string crs and $(x, w) \in R$, and outputs a proof π.
– The deterministic *verifier algorithm* Vrfy takes as input crs, a string x, and a proof π, and outputs a bit indicating acceptance or rejection.
– The *simulated parameter-generation algorithm* SimGen takes as input the security parameter 1^κ and outputs a common reference string crs and a trapdoor td.
– The *simulated prover algorithm* $\mathsf{SimProve}$ takes as input an instance x and a trapdoor td, and outputs a proof π.

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

NIZK is a non-interactive proof system if it satisfies soundness:

Definition 4. *For a tuple of algorithms* NIZK *as above and an algorithm* $\mathcal{A}$*, define experiment* $\mathbf{SND}^{\mathcal{A}}_{\mathsf{NIZK}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{GenZK}(1^\kappa)$.
2. *Run* $\mathcal{A}$ *on input* crs *to obtain* (x, π)*. The experiment evaluates to* 1 *iff* $\mathsf{Vrfy}(\mathsf{crs}, x, \pi) = 1$ *and* $x \notin \mathcal{L}_R$.

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

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

We say that NIZK is a $(t_p, t_v, t_{sgen}, t_{sp})$ non-interactive proof system (for relation R) if Prove runs in time t_p, Vrfy runs in time t_v, SimGen runs in time t_{sgen}, and $\mathsf{SimProve}$ runs in time t_{sp}.

We next define zero-knowledge and simulation soundness.

Definition 5. *For a tuple of algorithms* NIZK *as above and an algorithm* $\mathcal{A}$*, define experiment* $\mathbf{ZK}^{\mathcal{A}}_{\mathsf{NIZK}}$ *as follows:*

1. *Choose a uniform bit* $b \leftarrow \{0, 1\}$.
2. *Compute* $\mathsf{crs}_0 \leftarrow \mathsf{GenZK}(1^\kappa)$ *and* $(\mathsf{crs}_1, td) \leftarrow \mathsf{SimGen}(1^\kappa)$.
3. *Run* $\mathcal{A}$ *on input* crs_b *with access to a* prover oracle PROVE *that behaves as follows: on input* (x, w)*,* PROVE *returns* $\perp$ *if* $(x, w) \notin R$*; otherwise it computes* $\pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}_0, x, w)$ *and* $\pi_1 \leftarrow \mathsf{SimProve}(x, td)$*, and returns* π_b.
4. *When* $\mathcal{A}$ *outputs a bit* b'*, the experiment evaluates to* 1 *iff* $b' = b$.

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

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

Simulation soundness says that an adversary cannot produce a fake proof even if it has oracle access to the simulated prover algorithm.

Definition 6. *For a tuple of algorithms* NIZK *as above and an algorithm* $\mathcal{A}$*, define experiment* $\mathbf{SIMSND}^{\mathcal{A}}_{\mathsf{NIZK}}$ *as follows:*

1. *Compute* $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$.
2. *Run* $\mathcal{A}$ *on input* crs *with access to* $\mathsf{SimProve}(\cdot, t)$*. Let* $\mathcal{Q}$ *denote the set of queries* $\mathcal{A}$ *makes to this oracle.*
3. $\mathcal{A}$ *outputs* (x, π)*, and the experiment evaluates to* 1 *iff* $\mathsf{Vrfy}(\mathsf{crs}, x, \pi) = 1$*,* $x \notin \mathcal{L}_R$*, and* $x \notin \mathcal{Q}$*.*

NIZK *is* (t, ϵ)-simulation sound *if for all* $\mathcal{A}$ *running in time at most* t*,*

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

3 Algebraic Hardness of the RSW Problem

We briefly recall the AGM, and then introduce a refinement that we call the *strong AGM* (SAGM), which lies in between the GGM and the AGM. As the main result of this section, we show that the RSW assumption can be reduced to the factoring assumption in the strong AGM. For completeness, we also show that it is not possible to reduce hardness of RSW to hardness of factoring in the AGM (unless factoring is easy).

3.1 The Strong Algebraic Group Model

The *algebraic group model* (AGM), introduced by Fuchsbauer, Kiltz, and Loss [16], lies between the GGM and the standard model. As in the standard model, algorithms are given actual (bit-strings representing) group elements, rather than abstract handles for (or random encodings of) those elements, as in the GGM. This means that AGM algorithms are strictly more powerful than GGM algorithms (e.g., when working in $\mathbb{Z}_N^*$ an AGM algorithm can compute Jacobi symbols), and in particular means that the computational difficulty of problems in the AGM depends on the group representation used. (In contrast, in the GGM all cyclic groups of the same order are not only isomorphic, but are identical.) On the other hand, an algorithm in the AGM that outputs group elements must also output representations of those elements with respect to any inputs the algorithm has received; this restricts the algorithm in comparison to the standard model (which imposes no such restriction).

More formally, we define the notion of an *algebraic* algorithm [8, 27]:

Definition 7 (Algebraic algorithm). *An algorithm* $\mathcal{A}$ *is called* algebraic (over group $\mathbb{G}$) *if whenever* $\mathcal{A}$ *outputs a group element* $\mathbf{X} \in \mathbb{G}$*, it also outputs an integer vector* $\vec{\lambda}$ *with* $\mathbf{X} = \prod_i L_i^{\lambda_i}$*, where* $\vec{L}$ *denotes the (ordered) list of group elements that* $\mathcal{A}$ *has received as input up to that point. We refer to* $\vec{\lambda}$ *as an* algebraic representation *of* $\mathbf{X}$*.*

The original formulation of the AGM assumes that $\mathbb{G}$ is a group of (known) prime order but this is not essential and we do not make that assumption here.

The strong AGM. The AGM does not directly provide a way to measure the number of (algebraic) steps taken by an algorithm. This makes it unsuitable for dealing with "fine-grained" assumptions like the hardness of the RSW problem. (This point is made more formal in Section 3.3. On the other hand, as we will see, from a "coarse" perspective any algebraic algorithm can be implemented using polylogarithmically many algebraic steps.) This motivates us to consider a refinement of the AGM that we call the *strong AGM (SAGM)*, which provides a way to directly measure the number of group operations an algorithm performs.

In the AGM, whenever an algorithm outputs a group element $\mathbf{X}$ it is required to also provide an algebraic representation of $\mathbf{X}$ with respect to all the group elements the algorithm has received as input so far. In the SAGM we strengthen this and require an algorithm to express any group element as either (1) a *product* of two previous group elements that it has either received as input or already computed in some intermediate step, or (2) an *inverse* of a previous group element that it has either received as input or already computed in some intermediate step. That is, we require algorithms to be *strongly algebraic*:

Definition 8 (Strongly algebraic algorithm). *An algorithm $\mathcal{A}$ over $\mathbb{G}$ is called* strongly algebraic *if in each (algebraic) step $\mathcal{A}$ does arbitrary local computation and then outputs[2] one or more tuples of the following form:*

1. $(\mathbf{X}, \mathbf{X}_1, \mathbf{X}_2) \in \mathbb{G}^3$*, where* $\mathbf{X} = \mathbf{X}_1 \cdot \mathbf{X}_2$ *and* $\mathbf{X}_1, \mathbf{X}_2$ *were either provided as input to $\mathcal{A}$ or were output by $\mathcal{A}$ in some previous step(s);*
2. $(\mathbf{X}, \mathbf{X}_1) \in \mathbb{G}^2$*, where* $\mathbf{X} = \mathbf{X}_1^{-1}$ *and* $\mathbf{X}_1$ *was either provided as input to $\mathcal{A}$ or was output by $\mathcal{A}$ in some previous step.*

The running time of $\mathcal{A}$ is the number of algebraic steps it takes.

Note that we allow arbitrary parallelism, since we allow strongly algebraic algorithms to output multiple tuples per step.

As an example of a strongly algebraic algorithm, consider the following algorithm[3] $\widetilde{\mathsf{Mult}}$ computing the product of n input elements $\mathbf{X}_1, \ldots, \mathbf{X}_n$ 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 $(\mathbf{YZ}, \mathbf{Y}, \mathbf{Z})$. It is also easy to see that the repeated squaring algorithm RepSqr described previously can be cast as a strongly algebraic algorithm $\widetilde{\mathsf{RepSqr}}$ such that $\widetilde{\mathsf{RepSqr}}(g, x)$ computes g^x in $\lceil \log x \rceil$ steps.

Any algebraic algorithm with polynomial-length output can be turned into a strongly algebraic algorithm that uses polylogarithmically many steps:

[2] Here we require $\mathcal{A}$ to output group elements at intermediate steps of its computation. Technically, we can distinguish the final output of $\mathcal{A}$ by requiring $\mathcal{A}$ to output a special indicator when generating its final output.

[3] In general we use $\widetilde{\cdot}$ to indicate that an algorithm is strongly algebraic.

Theorem 1. *Let $\mathcal{A}$ be an algebraic algorithm over $\mathbb{G}$ taking as input n group elements $\mathbf{X}_1, \ldots, \mathbf{X}_n$ and outputting a group element $\mathbf{X}$ along with its algebraic representation $(\lambda_1, \ldots, \lambda_n)$ (so $\mathbf{X} = \mathbf{X}_1^{\lambda_1} \cdots \mathbf{X}_n^{\lambda_n}$), where $|\lambda_i| \leq 2^\kappa$. Then there is a strongly algebraic algorithm $\tilde{\mathcal{A}}$ over $\mathbb{G}$ running in $1 + \kappa + \lceil \log n \rceil$ steps whose final output is identically distributed.*

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

1. Run $\mathcal{A}(\mathbf{X}_1, \ldots, \mathbf{X}_n)$ and receive $\mathcal{A}$'s output $\mathbf{X}$ together with $(\lambda_1, \ldots, \lambda_n)$. (Note that this is an internal computation step, not not an algebraic step; in particular, no group element is being output by $\tilde{\mathcal{A}}$ here.)
2. For all i, let $\mathbf{X}'_i = \mathbf{X}_i^{-1}$. (Formally, $\tilde{\mathcal{A}}$ outputs $(\mathbf{X}'_i, \mathbf{X}_i)$ for all i. This can all be done in one algebraic step.)
3. For all i, if $\lambda_i \geq 0$ compute $\mathbf{X}_1^{\lambda_i} := \widetilde{\mathsf{RepSqr}}(\mathbf{X}_i, \lambda_i)$; if $\lambda_i < 0$ compute $\mathbf{X}_1^{\lambda_i} := \widetilde{\mathsf{RepSqr}}(\mathbf{X}'_i, |\lambda_i|)$. All these (algebraic) computations are done in parallel.
4. Run $\widetilde{\mathsf{Mult}}(\mathbf{X}_1^{\lambda_1}, \ldots, \mathbf{X}_n^{\lambda_n})$.

Running time in the SAGM. Definition 8 only accounts for the number of algebraic steps used by an algorithm. In some cases, we may also wish to account for other (non-algebraic) computation that an algorithm does, measured in some underlying computational model (e.g., the Turing machine model). In this case we will express the running time of algorithms as a *pair* and say that a strongly algebraic algorithm runs in time (t_1, t_2) if it uses t_1 algebraic steps, and has running time t_2 in the underlying computational model.

3.2 Hardness of the RSW Problem in the Strong AGM

If the factorization of N (and hence $\phi(N)$) is known, then $g^{2^T} \bmod N$ can be computed in at most $\lceil \log \phi(N)/4 \rceil$ algebraic steps by first computing $z := 2^T \bmod \phi(N)/4$ and then computing $\widetilde{\mathsf{RepSqr}}(g, z)$. Thus, informally, if the RSW problem is hard then factoring must be hard as well. Here we prove a converse in the SAGM, showing that the hardness of factoring implies the hardness of solving the T-RSW problem in fewer than T sequential steps for a strongly algebraic algorithm. We rely on a concrete version of the well-known result that N can be factored efficiently given any positive multiple of $\phi(N)$ (a proof follows from a quantitative analysis of the proof of [18, Theorem 8.50]):

Lemma 1. *There is an algorithm* Factor *running in time $4\lceil \log \alpha \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil$ that takes as input a modulus N (that is the product of two κ-bit safe primes) along with $m = \alpha \cdot \phi(N)$ for $\alpha \in \mathbb{Z}^+$, and outputs the factorization of N with probability at least $\frac{1}{2}$.*

We now show:

Theorem 2. *Assume factoring is $\left(t_p + t_o + \Theta(\kappa) + 4\lceil \log T \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil, \epsilon\right)$-hard relative to* GenMod*. Then the T-RSW problem is $\left((0, t_p), (T-1, t_o), 2\epsilon\right)$-hard relative to* GenMod *in the SAGM.*

Proof. Let $\mathcal{A}$ be a strongly algebraic algorithm that uses no algebraic steps and runs in time t_p in the preprocessing phase, and uses at most $T-1$ algebraic steps and runs in time t_o in the online phase. Let g be the generator given to $\mathcal{A}$ at the beginning of the online phase of T-$\mathbf{RSW}^{\mathcal{A}}_{\mathsf{GenMod}}$. For any $\mathbf{X} \in \mathbb{QR}_N$ output by $\mathcal{A}$ as part of an algebraic step during the online phase of T-$\mathbf{RSW}_{\mathsf{GenMod}}$, we recursively define $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) \in \mathbb{Z}^+$ as:

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

$$\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) = \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1) + \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2);$$

- If $\mathcal{A}$ outputs $(\mathbf{X}, \mathbf{X}_1)$ in an algebraic step, then $\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}) = -\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1)$.

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

Claim. For any strongly algebraic algorithm $\mathcal{A}$ given only g as input and running in $s \geq 1$ algebraic steps, every $\mathbf{X} \in \mathbb{QR}_N$ output by $\mathcal{A}$ satisfies $|\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})| \leq 2^s$.

Proof. The proof is by induction on s. If $s = 1$, the only group elements $\mathcal{A}$ can output are g^{-1} or g^2, so the claim holds. Suppose the claim holds for $s-1$. If $\mathcal{A}$ outputs $(\mathbf{X}, \mathbf{X}_1, \mathbf{X}_2)$ in step s, then $\mathbf{X}_1, \mathbf{X}_2$ must either be equal to g or have been output in a previous step. So the induction hypothesis tells us that $|\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1)|, |\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2)| \leq 2^{s-1}$. It follows that

$$|\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})| = |\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1) + \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2)| \leq |\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1)| + |\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_2)| \leq 2^s.$$

Similarly, if $\mathcal{A}$ outputs $(\mathbf{X}, \mathbf{X}_1)$ in step s, then $|\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})| = |\mathsf{DL}_{\mathcal{A}}(g, \mathbf{X}_1)| \leq 2^{s-1}$. In either case, the claim holds for s as well.

We construct an algorithm $\mathcal{R}$ that factors N as follows. $\mathcal{R}$, on input N, runs the preprocessing phase of $\mathcal{A}(N)$, and then samples $g \leftarrow \mathbb{QR}_N$ and runs the online phase of $\mathcal{A}(g)$. When $\mathcal{A}$ produces its final output $\mathbf{X}$, then $\mathcal{R}$ computes $x = \mathsf{DL}_{\mathcal{A}}(g, \mathbf{X})$. Finally, $\mathcal{R}$ sets $m := 4 \cdot (2^T - x)$ and outputs $\mathsf{Factor}(N, m)$.

When $\mathbf{X} = g^{2^T} \bmod N$ we have $x = 2^T \bmod \phi(N)/4$, i.e., $\phi(N)$ divides $m = 4 \cdot (2^T - x)$. By the previous claim, $|x| < 2^T$ and so m is a nontrivial (integer) multiple of $\phi(N)$. We thus see that $\mathcal{R}$ factors N with probability at least $\frac{1}{2} \cdot \Pr\left[T\text{-}\mathbf{RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right]$. The running time of $\mathcal{R}$ is at most $t_p + t_o + \Theta(\kappa) + 4\lceil \log T \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil$. This completes the proof.

3.3 The RSW Problem in the AGM

In the previous section we showed that the hardness of the RSW problem can be reduced to the hardness of factoring in the *strong* AGM. Here, we show that a similar reduction in the (plain) AGM is impossible, unless factoring is easy. Specifically, we give a "meta-reduction" $\mathcal{M}$ that converts any such reduction $\mathcal{R}$ into an efficient algorithm for factoring. In the theorem that follows, we write $\mathcal{R}^{\mathcal{A}}$ to denote execution of $\mathcal{R}$ given (black-box) oracle access to another algorithm $\mathcal{A}$. When we speak of the running time of $\mathcal{R}$ we assign unit cost to its oracle calls.

Theorem 3. *Let $\mathcal{R}$ be a reduction running in time t_R and such that for any algebraic algorithm $\mathcal{A}$ with* $\Pr\left[T\text{-}\mathbf{RSW}^{\mathcal{A}}_{\mathsf{GenMod}} = 1\right] = 1$, *algorithm $\mathcal{B} = \mathcal{R}^{\mathcal{A}}$ satisfies* $\Pr\left[\mathbf{FAC}^{\mathcal{B}}_{\mathsf{GenMod}} = 1\right] > \epsilon$. *Then there is an algorithm $\mathcal{M}$ running in time* $\mathsf{poly}(t_R)$ *with* $\Pr\left[\mathbf{FAC}^{\mathcal{M}}_{\mathsf{GenMod}} = 1\right] > \epsilon$.

Proof. Let $\mathcal{R}$ be as described in the theorem statement. $\mathcal{M}$ simply runs $\mathcal{R}$, handling its oracle calls by simulating the behavior of an (algebraic) algorithm $\mathcal{A}$ that solves the RSW problem with probability 1. (Note that the running time of doing so is irrelevant as far as the behavior of $\mathcal{R}$ is concerned, since $\mathcal{R}$ cannot observe the running time of $\mathcal{A}$. For this reason, we also ignore the fact that $\mathcal{A}$ is allowed preprocessing, and simply consider an algorithm $\mathcal{A}$ for which $\mathcal{A}(N, g)$ outputs $(g^{2^T} \bmod N, 2^T)$.) Formally, $\mathcal{M}(N)$ runs $\mathcal{R}(N)$; when $\mathcal{R}$ makes an oracle query $\mathcal{A}(N', g)$, algorithm $\mathcal{M}$ answers the query by computing $\mathbf{X} = g^{2^T} \bmod N'$ (using RepSqr) and returning $(\mathbf{X}, 2^T)$ to $\mathcal{R}$. Finally, $\mathcal{M}$ outputs the factors that are output by $\mathcal{R}$.

The assumptions of the theorem imply that $\mathcal{M}$ factors N with probability at least ϵ. The running time of $\mathcal{M}$ is the running time of $\mathcal{R}$ plus the time to run RepSqr each time $\mathcal{R}$ calls $\mathcal{A}$.

4 Non-Malleable Timed Commitments

Here we provide appropriate definitions for non-interactive (non-malleable) timed commitments (NITCs). As a building block toward our construction of NITCs, we introduce the notion of *timed public-key encryption* (TPKE) and show how to construct CCA-secure TPKE.

4.1 Definitions

Timed commitments allow a committer to commit to a message m such that binding holds as usual, but hiding holds only until some designated time T; the receiver can "force open" the commitment by that time. Boneh and Naor [7] gave a (somewhat informal) description of the syntax of *interactive* timed commitments along with some specific constructions. We introduce the syntax of *non-interactive* timed commitments and give appropriate security definitions.

Definition 9. *A $(t_{com}, t_{cv}, t_{dv}, t_{fo})$*-non-interactive timed commitment scheme (NITC) *is a tuple of algorithms* $\mathsf{TC} = (\mathsf{PGen}, \mathsf{Com}, \mathsf{ComVrfy}, \mathsf{DecVrfy}, \mathsf{FDecom})$ *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 $\pi_{\mathsf{Com}}, \pi_{\mathsf{Dec}}$ in time at most t_{com}.*

– *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 $\mathsf{DecVrfy}$ *takes as input a string* crs*, a commitment* C*, a message* m*, and a proof* π_{Dec}*. 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 require that for all κ*, all* crs *output by* $\mathsf{PGen}(1^\kappa)$*, all* m*, and all* $C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Dec}}$ *output by* $\mathsf{Com}(\mathsf{crs}, m)$*, it holds that*

$$\mathsf{ComVrfy}(\mathsf{crs}, C, \pi_{\mathsf{Com}}) = \mathsf{DecVrfy}(\mathsf{crs}, C, m, \pi_{\mathsf{Dec}}) = 1$$

and $\mathsf{FDecom}(\mathsf{crs}, C) = m$*.*

To commit to message m, the committer runs Com to get C, π_{Com}, and π_{Dec}, and sends C and π_{Com} to a receiver. The receiver can run $\mathsf{ComVrfy}$ to check that C can be forcibly decommitted (if need be). To decommit, the committer sends m and π_{Dec} to the receiver, who can then run $\mathsf{DecVrfy}$ to verify the claimed opening. If the committer refuses to decommit, C be opened using FDecom. NITCs are generally only interesting when $t_{cv}, t_{dv} \ll t_{fo}$, i.e., when forced opening of a commitment takes longer than verification.

NITCs must satisfy appropriate notions of hiding and binding. For hiding, we use a strong definition based on *non-malleability* that is modeled on the CCA-security notion for (standard) commitments given by Canetti et al. [10]. Specifically, we require hiding to hold even when the adversary is given access to an oracle that provides the forced openings of commitments of the adversary's choice. In the timed setting that we are considering, the motivation behind providing the adversary with such an oracle is that (honest) parties may be running machines that can force open commitments at different speeds. As such, the adversary (as part of some higher-level protocol) could trick a party into opening commitments of the attacker's choice. Note that although the adversary could run the forced opening algorithm itself, doing so would incur a significant computational cost; in contrast, each of the adversary's queries to the forced-opening oracle incurs only unit cost.

Definition 10. *For an NITC scheme* TC *and algorithm* $\mathcal{A}$*, define experiment* $\textbf{IND-CCA}^{\mathcal{A}}_{\mathsf{TC}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{PGen}(1^\kappa)$*.*
2. *Run* $\mathcal{A}(\mathsf{crs})$ *in a preprocessing phase with access to* $\mathsf{FDecom}(\mathsf{crs}, \cdot)$*.*
3. *When* $\mathcal{A}$ *outputs* (m_0, m_1)*, choose a uniform bit* $b \leftarrow \{0, 1\}$ *and then compute* $(C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Dec}}) \leftarrow \mathsf{Com}(\mathsf{crs}, m_b)$*. Give* (C, π_{Com}) *to* $\mathcal{A}$*, who continues to have access to* $\mathsf{FDecom}(\mathsf{crs}, \cdot)$ *except that it may not query the oracle on* C*.*
4. *When* $\mathcal{A}$ *outputs a bit* b'*, the experiment evaluates to* 1 *iff* $b' = b$*.*

TC *is* (t_p, t_o, ϵ)-CCA-secure *if for all adversaries* $\mathcal{A}$ *running in time at most* t_p *in the preprocessing phase and time at most* t_o *in the subsequent online phase,*

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

Binding. The binding property states that a valid commitment cannot be opened to two different messages, and will be forced open to the correct message. We require this to hold even in the presence of a forced-opening oracle as before.

Definition 11. *For an NITC scheme* TC *and algorithm* $\mathcal{A}$*, define experiment* $\mathbf{BND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TC}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{PGen}(1^\kappa)$.
2. *Run* $\mathcal{A}(\mathsf{crs})$ *with access to* $\mathsf{FDecom}(\mathsf{crs}, \cdot)$.
3. $\mathcal{A}$ *outputs* $(m, C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Dec}}, m', \pi'_{\mathsf{Dec}})$*, and the experiment evaluates to* 1 *iff* $\mathsf{ComVrfy}(\mathsf{crs}, C, \pi_{\mathsf{Com}}) = 1$ *and either:*
   - $m' \neq m$*, yet* $\mathsf{DecVrfy}(\mathsf{crs}, C, m, \pi_{\mathsf{Dec}}) = \mathsf{DecVrfy}(\mathsf{crs}, C, m', \pi'_{\mathsf{Dec}}) = 1$*;*
   - $\mathsf{DecVrfy}(\mathsf{crs}, C, m, \pi_{\mathsf{Dec}}) = 1$ *but* $\mathsf{FDecom}(\mathsf{crs}, C) \neq m$.

TC *is* (t, ϵ)-BND-CCA-secure *if for all adversaries* $\mathcal{A}$ *running in time* t*,*

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

Timed public-key encryption. TPKE can be viewed as the counterpart of timed commitments for public-key encryption. As in the case of standard public-key encryption (PKE), a sender encrypts a message for a designated recipient using the recipient's public key; that recipient can decrypt and recover the message. *Timed* PKE additionally allows anyone (and not just the sender) to recover the message, but only by investing more computational effort.

Definition 12. *A* (t_e, t_{fd}, t_{sd})-timed public-key encryption (TPKE) scheme *is a tuple of algorithms* $\mathsf{TPKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *such that:*

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

We require that for all κ*, all* (pk, sk) *output by* $\mathsf{KGen}(1^\kappa)$*, all* m*, and all* c *output by* $\mathsf{Enc}(pk, m)$*, it holds that* $\mathsf{Dec}_f(sk, c) = \mathsf{Dec}_s(pk, c) = m$.

TPKE schemes are only interesting when $t_{fd} \ll t_{sd}$, i.e., when fast decryption is much faster than slow decryption.

We consider security of TPKE against chosen-ciphertext attacks.

Definition 13. *For a TPKE scheme* TPKE *and algorithm* $\mathcal{A}$*, define experiment* $\mathbf{IND\text{-}CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ *as follows:*

1. *Compute* $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$.
2. *Run* $\mathcal{A}(pk)$ *with access to* $\mathsf{Dec}_f(sk, \cdot)$ *in a preprocessing phase.*
3. *When* $\mathcal{A}$ *outputs* (m_0, m_1)*, choose* $b \leftarrow \{0, 1\}$*, compute* $c \leftarrow \mathsf{Enc}(pk, m_b)$*, and run* $\mathcal{A}(c)$ *in the online phase.* $\mathcal{A}$ *continues to have access to* $\mathsf{Dec}_f(sk, \cdot)$*, except that* $\mathcal{A}$ *may not query this oracle on* c.
4. *When* $\mathcal{A}$ *outputs a bit* b'*, the experiment evaluates to* 1 *iff* $b' = b$.

TPKE *is* (t_p, t_o, ϵ)-CCA-secure *iff for all* $\mathcal{A}$ *running in time at most* t_p *in the preprocessing phase and running in time at most* t_o *in the online phase,*

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

We remark that for some applications of TPKE, one might also want to consider a "binding" property requiring that fast and slow decryption return the same result even for maliciously generated cpihertexts; one can verify that this property is satisfied by our TPKE scheme in the next section. However, since our primary motivation for introducing TPKE is to construct NTICs, we do not require this notion here.

4.2 CCA-Secure TPKE

We describe a construction of a TPKE scheme that is CCA-secure under the decisional RSW assumption. While our construction is in the standard model, it suffers from a slow encryption algorithm. In the full version of our paper, we describe a CCA-secure TPKE scheme in the random-oracle model where encryption can be done more quickly with knowledge of the secret key.

The starting point of our construction is a CPA-secure TPKE scheme based on the decisional RSW assumption. In this scheme, the public key is a modulus N and a generator $g \in \mathbb{QR}_N$; the secret key includes $\phi(N)$. To encrypt a message[4] $\mathbf{M} \in \mathbb{QR}_N$, the sender chooses a random generator $\mathbf{R}$ (by raising g to a random power modulo N), and computes the ciphertext $(\mathbf{R},\ \mathbf{R}^{2^T} \cdot \mathbf{M} \bmod N)$. This ciphertext can be decrypted quickly using $\phi(N)$, but can also be decrypted slowly without knowledge of $\phi(N)$.

We can obtain a CCA-secure TPKE scheme by suitably adapting the Naor-Yung paradigm [26, 31] to the setting of timed encryption. The Naor-Yung approach constructs a CCA-secure encryption scheme by encrypting a message

[4] This can be extended to encryption of a message $m < \sqrt{N}$ by encoding m as $\mathbf{M} := m^2 \in \mathbb{QR}_N$; note that m can be recovered from $\mathbf{M}$ by computing the square root of $\mathbf{M}$ over the integers.

For fixed integer T, define the relation

$$R = \left\{ ((\mathbf{R}_1, \mathbf{R}_2, \mathbf{X}_1, \mathbf{X}_2, N_1, N_2), \mathbf{M}) \mid \bigwedge_{i=1,2} \mathbf{X}_i = \mathbf{R}_i^{2^T} \cdot \mathbf{M} \bmod N_i \right\}.$$

(We drop N_1, N_2 from the instance when they are clear from context.) Let $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ be an NIZK proof system for R. Define a TPKE scheme (parameterized by T) as follows:

- $\mathsf{KGen}(1^\kappa)$: For $i \in \{1, 2\}$ compute $(N_i, p_i, q_i) \leftarrow \mathsf{GenMod}(1^\kappa)$, set $\phi_i := \phi(N_i)$, set $z_i := 2^T \bmod \phi_i$, and choose $g_i \leftarrow \mathbb{QR}_{N_i}$. Run $\mathsf{crs} \leftarrow \mathsf{GenZK}(1^\kappa)$. Output $pk := (\mathsf{crs}, N_1, N_2, g_1, g_2)$ and $sk := (\mathsf{crs}, N_1, z_1)$.
- $\mathsf{Enc}((\mathsf{crs}, N_1, N_2, g_1, g_2), \mathbf{M})$: For $i = 1, 2$, choose $r_i \leftarrow \mathbb{Z}_{N_i^2}$ and compute

  $$\mathbf{R}_i := g_i^{r_i} \bmod N_i, \quad \mathbf{Z}_i := \mathbf{R}_i^{2^T} \bmod N_i, \quad \mathbf{C}_i := \mathbf{Z}_i \cdot \mathbf{M} \bmod N_i,$$

  using RepSqr. Also compute $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \mathbf{M})$. Output the ciphertext $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$.
- $\mathsf{Dec}_f(sk, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: If $\mathsf{Vrfy}(\mathsf{crs}, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 0$, output $\perp$. Else compute $\mathbf{Z}_1 := \mathbf{R}_1^{z_1} \bmod N_1$ (using RepSqr) and then output $\mathbf{M} := \mathbf{C}_1 \mathbf{Z}_1^{-1} \bmod N_1$.
- $\mathsf{Dec}_s(pk, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: If $\mathsf{Vrfy}(\mathsf{crs}, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 0$, output $\perp$. Else compute $\mathbf{Z}_1 := \mathbf{R}_1^{2^T} \bmod N_1$ (using RepSqr) and then output $\mathbf{M} := \mathbf{C}_1 \mathbf{Z}_1^{-1} \bmod N_1$.

Fig. 1. A CCA-secure TPKE scheme.

twice using independent instances of a CPA-secure encryption scheme accompanied by a simulation-sound NIZK proof showing that the two messages encrypted are the same. In our setting, we need the NIZK proof system to also have "fast" verification and simulation (specifically, linear in the size of the input instance). We present the details of our construction in Figure 1.

The proof of security in our context requires the ability to simulate the decryption oracle using a "fast" decryption algorithm. The reason is that if it were not possible to simulate decryption quickly, then the reduction to the decisional RSW assumption would take too much time. Fast simulation for our scheme is possible since in the proof of security for the Naor-Yung construction, the simulator knows (at least) one of the secret keys at any time; the corresponding key can be used to decrypt quickly.

Theorem 4. *Fix t_p, t_o, and $t = t_p + t_o$, and let NIZK be a $(t_{pr}, t_v, t_{sgen}, t_{sp})$-proof system. If NIZK is $(t \cdot (t_v + \Theta(\kappa)), \epsilon_{ZK})$-zero-knowledge and $(\Theta(t \cdot \kappa), \epsilon_{SS})$-simulation sound, and the decisional T-RSW problem is*

$$(t_{sgen} + t_p \cdot (t_v + \Theta(\kappa)),\ t_{sp} + t_o \cdot (t_v + \Theta(\kappa)),\ \epsilon)\text{-hard}$$

relative to GenMod*, then the TPKE scheme in Figure 1 is* $(t_p, t_o, \epsilon_{ZK} + \epsilon_{SS} + 2\epsilon)$*-CCA-secure.*

Proof. Let $\mathcal{A}$ be an adversary with preprocessing time t_p and online time t_o, and note that the total number of decryption-oracle queries made by $\mathcal{A}$ is at most $t = t_p + t_o$. We define a sequence of experiments as follows.

Expt_0: This is the original CCA-security experiment $\mathbf{IND\text{-}CCA}_{\mathsf{TPKE}}$. Denote $\mathcal{A}$'s challenge ciphertext by $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$.

Expt_1: Expt_1 is identical to Expt_0, except that crs and π^* are simulated. That is, $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$ is computed as part of key generation and crs is included in the public key; the challenge ciphertext is computed as before, except that the proof is now computed as $\pi^* \leftarrow \mathsf{SimProve}((\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*), td)$.

We upper bound $|\Pr[\mathsf{Expt}_1^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_0^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{ZK}$ to the zero-knowledge property of NIZK. Reduction $\mathcal{R}_{ZK}$ is given crs, which it includes as part of the public key. (The rest of the key is generated as in Expt_0 and Expt_1.) When $\mathcal{R}_{ZK}$ computes the challenge ciphertext, it uses its PROVE oracle to generate π^*. $\mathcal{R}_{ZK}$ runs in time $t \cdot (t_v + \Theta(\kappa))$, and thus

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

Expt_2: In Expt_2 component $\mathbf{C}_2^*$ of the challenge ciphertext is computed by choosing $u_2 \leftarrow \mathbb{Z}_{N_2^2}$, computing $\mathbf{U}_2 := \mathsf{RepSqr}(g_2, N_2, u_2)$, and then setting $\mathbf{C}_2^* := \mathbf{U}_2 \cdot \mathbf{M}_b \bmod N_2$. We bound $|\Pr[\mathsf{Expt}_2^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_1^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{DRSW}$ to the decisional T-RSW problem that works as follows:

- Preprocessing phase: $\mathcal{R}_{DRSW}$, on input N_2, runs $(N_1, p_1, q_1) \leftarrow \mathsf{GenMod}(1^\kappa)$, computes $\phi_1 := \phi(N_1) = (p_1 - 1)(q_1 - 1)$, sets $z_1 := 2^T \bmod \phi_1$, and chooses $g_1 \leftarrow \mathbb{QR}_{N_1}$, $g_2 \leftarrow \mathbb{QR}_{N_2}$; it also runs $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$. Then $\mathcal{R}_{DRSW}$ runs $\mathcal{A}(\mathsf{crs}, N_1, N, g_1, g_2)$, answering the decryption-oracle queries of $\mathcal{A}$ using the fast decryption algorithm (note $\mathcal{R}_{DRSW}$ knows z_1) until $\mathcal{A}$ makes its challenge query $(\mathbf{M}_0, \mathbf{M}_1)$.
- Online phase: $\mathcal{R}_{DRSW}$ is given $(\mathbf{R}_2^*, \mathbf{X}^*)$. It then chooses $b \leftarrow \{0, 1\}$ and $r_1 \leftarrow \mathbb{Z}_{N_1^2}$, and computes

  $$\mathbf{R}_1^* := \mathsf{RepSqr}(g_1, N_1, r_1), \quad \mathbf{Z}_1^* := \mathsf{RepSqr}(\mathbf{R}_1^*, N_1, z_1), \quad \mathbf{C}_1^* := \mathbf{Z}_1^* \cdot \mathbf{M}_b \bmod N_1,$$

  and

  $$\pi^* \leftarrow \mathsf{SimProve}((\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{X}^* \cdot \mathbf{M}_b), td).$$

  It returns $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{X}^* \cdot \mathbf{M}_b, \pi^*)$ to $\mathcal{A}$. It continues to answer decryption-oracle queries as before.
- When $\mathcal{A}$ outputs b', then $\mathcal{R}_{DRSW}$ outputs 1 iff $b' = b$.

$\mathcal{R}_{DRSW}$ runs in time $t_{sgen} + t_p \cdot (t_v + \Theta(\kappa))$ in the preprocessing phase and time $t_{sp} + t_o \cdot (t_v + \Theta(\kappa))$ in the online phase; thus,

$$|\Pr[\mathsf{Expt}_2^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_1^{\mathcal{A}} = 1]| \leq \epsilon.$$

Expt_3: Expt_3 is identical to Expt_2, except that $\mathbf{C}_2^*$ is computed as $\mathbf{U}_2$ (instead of $\mathbf{U}_2 \cdot \mathbf{M}_b$). Since the distributions of $\mathbf{U}_2$ and $\mathbf{U}_2 \cdot \mathbf{M}_b$ are both uniform, this is merely a syntactic change and $\Pr[\mathsf{Expt}_3^{\mathcal{A}} = 1] = \Pr[\mathsf{Expt}_2^{\mathcal{A}} = 1]$.

Expt_4: Expt_4 is identical to Expt_3, except that decryption-oracle queries are answered using $\mathbf{R}_2$ (instead of $\mathbf{R}_1$) to decrypt. That is, when $\mathcal{A}$ queries the decryption oracle on a ciphertext $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, then (assuming the proof verifies) we compute $\mathbf{Z}_2 := \mathsf{RepSqr}(\mathbf{R}_2, N_2, z_2)$ and return $\mathbf{M} := \mathbf{C}_2 \cdot \mathbf{Z}_2^{-1} \bmod N_2$.

Expt_4 and Expt_3 are identical unless $\mathcal{A}$ makes a decryption-oracle query $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$ for which the proof verifies yet $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1 \neq \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$. Denote this event Fake. We upper bound $\Pr[\mathsf{Fake}]$ by constructing a reduction $\mathcal{R}_{SS}$ to the simulation soundness of NIZK:

- $\mathcal{R}_{SS}(\mathsf{crs})$ does as follows: for $i = 1, 2$, it runs $(N_i, p_i, q_i) \leftarrow \mathsf{GenMod}(1^\kappa)$, computes $\phi_i := \phi(N_i) = (p_i - 1)(q_i - 1)$, sets $z_i := 2^T \bmod \phi_i$, and chooses $g_i \leftarrow \mathbb{QR}_{N_i}$. Then $\mathcal{R}_{SS}$ runs $\mathcal{A}(N, g, \mathsf{crs})$.
- When $\mathcal{A}$ queries the decryption oracle on ciphertext $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, then $\mathcal{R}_{SS}$ returns $\perp$ if $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 0$. Otherwise, $\mathcal{R}_{SS}$ checks if $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{z_1}} \bmod N_1 = \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{z_2}} \bmod N_2$ and, if so, returns $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{z_1}} \bmod N_1$, otherwise it outputs $((\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi)$ (and halts).
- When $\mathcal{A}$ makes its challenge query $(\mathbf{M}_0, \mathbf{M}_1)$, then $\mathcal{R}_{SS}$ chooses $b \leftarrow \{0, 1\}$ and computes

$$\mathbf{R}_1^* := \mathsf{RepSqr}(g_1, N_1, r_1), \quad \mathbf{Z}_1^* := \mathsf{RepSqr}(\mathbf{R}_1^*, N_1, z_1), \quad \mathbf{C}_1^* := \mathbf{Z}_1^* \cdot \mathbf{M}_b \bmod N_1,$$

$$u_2 \leftarrow \mathbb{Z}_{N_2^2}, \mathbf{C}_2^* := \mathsf{RepSqr}(g_2, N_2, u_2);$$

  it then obtains a (simulated) proof π^* using its oracle access to $\mathsf{SimProve}$. Finally, it gives the challenge ciphertext $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$ to $\mathcal{A}$. Subsequent decryption-oracle queries by $\mathcal{A}$ are answered as before.

$\mathcal{R}_{SS}$ runs in time $\Theta(t \cdot \kappa)$, succeeds exactly when Fake occurs, and simulates Expt_4 perfectly until Fake occurs. It follows that

$$|\Pr[\mathsf{Expt}_4^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_3^{\mathcal{A}} = 1]| \leq \Pr[\mathsf{Fake}] \leq \epsilon_{SS}.$$

Expt_5: Expt_5 is identical to Expt_4, except that $\mathbf{C}_1^*$ is set to a uniform value in $\mathbb{Z}_{N_1}$. The change here is symmetric to the one from Expt_1 to Expt_3, and so

$$|\Pr[\mathsf{Expt}_5^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_4^{\mathcal{A}} = 1]| \leq \epsilon.$$

Since b is independent of $\mathcal{A}$'s view in Expt_5, the probability that the CCA-experiment evaluates to 1 in this case is 1/2. We thus conclude that

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

which completes the proof.

We remark that for the preceding theorem to be meaningful, the decisional T-RSW problem should plausibly be hard for the concrete parameters stated; at a minimum, we require $t_{sp} + t_o \cdot (t_v + \Theta(\kappa)) < T$. This can be achieved by using a simulation-sound NIZK proof system for which verification and simulation can be done in time polynomial in the length of the witness, independent of the size of the circuit for the corresponding NP relation. In particular, suitable schemes [17, 20] with running times linear in the size of the witness can be used.

4.3 Constructing Non-Malleable Timed Commitments

In this section, we show how CCA-secure TPKE can be used to construct non-malleable timed commitments. The idea is very simple. The parameter-generation algorithm generates the keys for a timed public-key encryption scheme TPKE along with a common reference string for non-interactive zero-knowledge proof systems NIZK and NIZK'. To commit to a message m, the committer computes $c := \mathsf{Enc}(pk, m; r)$ (for some random coins r) and then (1) uses NIZK to prove that it knows (m, r) such that $c = \mathsf{Enc}(pk, m; r)$, and (2) uses NIZK' to prove that it knows r such that $c = \mathsf{Enc}(pk, m; r)$. The first of these proofs will be used as π_{Com}, i.e., to prove that the commitment is well-formed; the second proof will be used as π_{Dec}, i.e., to prove that m is the committed value. Details of our construction are presented in Figure 2. Note that for the construction to be meaningful we require the time needed for forced decommitment (i.e., the time for slow decryption in the underlying TPKE scheme) to be larger than the time required for proof verification. This can be satisfied when instantiating the NIZK proofs as discussed in the previous section.

Correctness of the scheme in Figure 2 follows immediately; we next show its CCA-security.

Theorem 5. *Fix $t_p.t_o$, and suppose TPKE is $(t_p + t_{sgen}, t_{sp}, \epsilon)$-CCA-secure, and $\mathsf{NIZK}, \mathsf{NIZK}'$ are $(t_p + t_o + t_e, \epsilon_{ZK})$-zero-knowledge. Then the NITC scheme in Figure 2 is $(t_p, t_o, \epsilon_{ZK} + \epsilon)$-CCA-secure.*

Proof. Let $\mathcal{A}$ be an adversary with preprocessing time t_p and online time t_o. Suppose $\mathcal{A}$'s challenge is (c^*, π^*). We define a sequence of experiments as follows.

Expt_0: This is the original CCA-security experiment $\mathbf{IND\text{-}CCA}_{\mathsf{TC}}$.

Expt_1: Expt_1 is identical to Expt_0, except that crs and π^* are simulated. That is, in the setup phase run $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$, and in the online phase compute $\pi^* \leftarrow \mathsf{SimProve}(c^*, td)$.

We upper bound $|\Pr[\mathsf{Expt}_1^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_0^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{ZK}$ to the zero-knowledge property of NIZK. $\mathcal{R}_{ZK}$ runs the code of Expt_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:

Let $\mathsf{TPKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ be a (t_e, t_{fd}, t_{sd})-TPKE scheme, $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ be a $(t_p, t_v, t_{sgen}, t_{sp})$-NIZK for relation

$$R = \{(c, (m, r)) \mid c = \mathsf{Enc}(pk, m; r)\},$$

and $\mathsf{NIZK}' = (\mathsf{GenZK}', \mathsf{Prove}', \mathsf{Vrfy}', \mathsf{SimGen}', \mathsf{SimProve}')$ be a $(t'_p, t'_v, t'_{sgen}, t'_{sp})$-NIZK for relation

$$R' = \{((c, m), r) \mid c = \mathsf{Enc}(pk, m; r)\}.$$

Define an NITC scheme as follows:

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

Fig. 2. An $(t_e + \max\{t_p, t'_p\}, t_v, t'_v, t_{sd})$-NITC scheme.

- Setup: $\mathcal{R}_{ZK}$, on input crs^*, runs $P \leftarrow \mathsf{PGen}(1^\kappa)$, $(sk, pk) \leftarrow \mathsf{KGen}(P)$ and $\mathsf{crs}' \leftarrow \mathsf{GenZK}'(1^\kappa)$, and runs $\mathcal{A}(pk, \mathsf{crs}^*, \mathsf{crs}')$. When $\mathcal{A}$'s makes a decryption-oracle query, $\mathcal{R}_{ZK}$ answers it using $\mathsf{Dec}_s(sk, \cdot)$.
- Online phase: When $\mathcal{A}$ makes its challenge query (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 decryption-oracle queries as before.
- Output: When $\mathcal{A}$ outputs b', $\mathcal{R}_{ZK}$ outputs 1 iff $b' = b$.

$\mathcal{R}_{ZK}$ runs in time $t_p + t_o + t_e$, and hence

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

Next we analyze $\mathcal{A}$'s advantage in Expt_1. Since the challenge is (c, π) where $c = \mathsf{Enc}(pk, m; r)$ and π is simulated without knowledge of m or r, $\mathcal{A}$'s advantage can be upper bounded directly by the CCA-security of TPKE. Formally, we upper bound $\mathcal{A}$'s advantage by constructing a reduction $\mathcal{R}_{CCA}$ to the CCA-security of TPKE (where $\mathcal{R}_{CCA}$'s decryption oracle is denoted $\mathsf{DEC}_{\mathsf{TPKE}}$):

- Preprocessing phase: $\mathcal{R}_{CCA}$, on input pk, computes $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$, and runs $\mathcal{A}(\mathsf{crs})$. When $\mathcal{A}$ makes a decryption-oracle query, $\mathcal{R}_{CCA}$ answers it using its own decryption oracle.
- Challenge query: When $\mathcal{A}$ outputs (m_0, m_1), $\mathcal{R}_{CCA}$ outputs the same messages. Given challenge ciphertext c^*, $\mathcal{R}_{CCA}$ computes $\pi^* \leftarrow \mathsf{SimProve}(c^*, td)$

and sends (c^*, π^*) to $\mathcal{A}$. After that, $\mathcal{R}$ answers $\mathcal{A}$'s decryption-oracle queries as before.
- Output: When $\mathcal{A}$ outputs a bit b', $\mathcal{R}_{CCA}$ also outputs b'.

$\mathcal{R}_{CCA}$ runs in time at most $t_p + t_{sgen}$ in the preprocessing phase, and time at most $t_o + t_{sp}$ in the online phase. $\mathcal{R}_{CCA}$ simulates Expt_1 perfectly, and wins if $\mathcal{A}$ wins. It follows that

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

We conclude that

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

which completes the proof.

We briefly sketch why our scheme satisfies our notion of binding. Recall that if $\mathcal{A}$ can win $\mathbf{BND\text{-}CCA}_{\mathsf{TC}}$, then it can produce a commitment c along with messages m, m' and proofs $\pi_{\mathsf{Com}}, \pi_{\mathsf{Dec}}$ such that $\mathsf{ComVrfy}((pk, \mathsf{crs}, \mathsf{crs}'), c, \pi_{\mathsf{Com}}) = \mathsf{DecVrfy}((pk, \mathsf{crs}, \mathsf{crs}'), c, m, \pi_{\mathsf{Dec}}) = 1$, $m' \neq m$, and either

$$(1) : \mathsf{FDecom}((pk, \mathsf{crs}, \mathsf{crs}'), c) = m'$$

or

$$(2) : \mathsf{DecVrfy}((pk, \mathsf{crs}, \mathsf{crs}'), c, m', \pi'_{\mathsf{Dec}}) = 1.$$

Soundness of NIZK and NIZK' implies that this is not possible.

References

1. D. Aggarwal and U. Maurer. Breaking RSA generically is equivalent to factoring. In *Advances in Cryptology—Eurocrypt 2009*, LNCS, pages 36–53. Springer, 2009.
2. C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner. Tardis: Time and relative delays in simulation. Cryptology ePrint Archive: Report 2020/537, 2020.
3. M. Bellare and S. Goldwasser. Verifiable partial key escrow. In *ACM Conf. on Computer and Communications Security (CCS) 1997*, pages 78–91. ACM Press, 1997.
4. N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In *ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356. ACM, 2016.
5. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *Advances in Cryptology—Crypto 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, 2018.
6. 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`.
7. D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology—Crypto 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, 2000.

8. D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology—Eurocrypt 1998*, LNCS, pages 59–71. Springer, 1998.
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.
10. 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 (FOCS)*, pages 541–550. IEEE, 2010.
11. J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and non-interactive timed-release encryption. In *7th Intl. Conf. on Information and Communication Security (ICICS)*, LNCS, pages 291–303. Springer, 2005.
12. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology—Crypto 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, 2001.
13. G. Di Crescenzo, R. Ostrovsky, and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Advances in Cryptology—Eurocrypt 1999*, LNCS, pages 74–89. Springer, 1999.
14. C. Dwork and M. Naor. Zaps and their applications. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 283–293. IEEE, 2000.
15. N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive: Report 2020/779, 2020.
16. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *Advances in Cryptology—Crypto 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, 2018.
17. J. Groth and M. Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In *Advances in Cryptology—Crypto 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, 2017.
18. J. Katz and Y. Lindell. *Introduction to Modern Cryptography (2nd edition)*. Chapman & Hall/CRC Press, 2014.
19. H. Lin, R. Pass, and P. Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In *58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 576–587. IEEE, 2017.
20. H. Lipmaa. Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, Report 2019/612, 2019. `https://eprint.iacr.org/2019/612`.
21. J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86:2549–2586, 2018.
22. M. Mahmoody, T. Moran, and S. P. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology—Crypto 2011*, volume 6841 of *LNCS*, pages 39–50. Springer, 2011.
23. G. Malavolta and S. A. K. Thyagarajan. Homomorphic time-lock puzzles and applications. In *Advances in Cryptology—Crypto 2019, Part I*, volume 11692 of *LNCS*, pages 620–649. Springer, 2019.
24. U. M. Maurer. Abstract models of computation in cryptography. In *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, 2005.
25. T. May. Timed-release crypto. `http://cypherpunks.venona.com/date/1993/02/msg00129.html`, 1993.
26. M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 427–437. ACM Press, 1990.

27. P. Paillier and D. Vergnaud. Discrete-log-based signatures may not be equivalent to discrete log. In *Advances in Cryptology—Asiacrypt 2005*, LNCS, pages 1–20. Springer, 2005.
28. K. Pietrzak. Simple verifiable delay functions. In *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15. Leibniz International Proceedings in Informatics (LIPIcs), 2019.
29. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, MIT Laboratory for Computer Science, 1996.
30. L. Rotem and G. Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In *Advances in Cryptology—Crypto 2020, Part III*, volume 12172 of *LNCS*, pages 481–509. Springer, 2020.
31. A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 543–553. IEEE, 1999.
32. V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—Eurocrypt 1997*, LNCS, pages 256–266. Springer, 1997.
33. B. Wesolowski. Efficient verifiable delay functions. In *Advances in Cryptology—Eurocrypt 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, 2019.