On the Security of Time-Lock Puzzles and Timed Commitments

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

[1] University of Maryland, College Park, USA
[2] George Mason University, Fairfax, USA
jkatz2@gmail.com, lossjulian@gmail.com, jiayux@uci.edu

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 of computation. Namely, in a quantitative version of the algebraic group model (AGM) that we call the *strong* AGM, we show that 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-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 "encrypt a message m into the future" such that m remains computationally hidden for some time T, but can be recovered once this time 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 [13], 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 the first formal evidence in support of the hardness of the most widely used time-lock puzzle [29] and (2) give new, stronger security definitions (and constructions) for timed commitments and related primitives. These contributions are explained in more detail next.

J. Katz—Portions of this work were done while at George Mason University.

© International Association for Cryptologic Research 2020
R. Pass and K. Pietrzak (Eds.): TCC 2020, LNCS 12552, pp. 390–413, 2020.
https://doi.org/10.1007/978-3-030-64381-2_14

Hardness in the (strong) AGM. The hardness assumption underlying the most popular time-lock puzzle [29] is that, given a random 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) [15] that we call the *strong AGM (SAGM)* that lies in between the generic group model (GGM) [24,32] 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 coefficients showing how x 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 output x. We show that if $\mathbb{QR}_N$ is modeled as a strongly algebraic group, then computing $g^{2^T} \bmod N$ from g using fewer than T squarings is as hard as factoring N. Our result is the first formal argument supporting the sequential hardness of squaring in $\mathbb{QR}_N$, and immediately implies the security of Pietrzak's VDF [28] in the SAGM (assuming the hardness of factoring). Our technique deviates substantially from known proofs in the AGM, which use groups of (known) prime order. We also show that in the AGM, it is not possible to reduce the hardness of speeding up sequential squaring to factoring (assuming factoring is hard in the first place).

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 works not already cited earlier. Mahmoody et al. [22] show constructions of time-lock puzzles in the random-oracle model, and Bitansky 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

[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].

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 [11,12] as well. Liu et al. [21] propose a time-released 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]; universally composable 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. [14] also 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 also give a generic construction of a time-lock puzzle from a VDF in the random-oracle model. Finally, the work of Rotem and Segev [30] analyzes the hardness of speeding up sequential squaring and related functions over the ring $\mathbb{Z}_N$. Their analysis is in the *generic ring model* [1], where an algorithm can only perform additions and multiplications modulo N, but the algorithm does not get access to the actual representations of ring elements. This makes their analysis incomparable to our analysis in the strong AGM.

1.2 Overview of the Paper

We introduce notation and basic definitions in Sect. 2. In Sect. 3 we introduce the SAGM and state our hardness result about the sequential squaring assumption. We give definitions for TPKE and NITC in Sect. 2, and give a construction of CCA-secure TPKE in Sect. 4.2. In Sect. 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.

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., 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. *When $\mathcal{A}$ outputs integers $p', q' \notin \{1, N\}$, 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 $g^1, \ldots, g^{2^i}$ (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 in the presence of 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 some intermediate state.*
3. *Sample $g \leftarrow \mathbb{QR}_N$ and run $\mathcal{A}$ on input g in the online phase.*
4. *When $\mathcal{A}$ outputs $\mathbf{X} \in \mathbb{QR}_N$, 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 there is a threshold $t^* \approx T$ such that the T-RSW problem is easy when $t_o \geq t^*$. In Sect. 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 way to compute $g^{2^T} \bmod N$ (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-$\mathbf{DRSW}_{\mathsf{GenMod}}$ 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 some 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. *When $\mathcal{A}$ outputs a bit b', 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 algorithms $\mathcal{A}$ 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.

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

Definition 4. *Let $\mathcal{L}_R$ be a language in NP defined by relation R. A* $(t_p, t_v, t_{sgen}, t_{sp})$-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_p 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_v 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_{sp}.*

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

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

Definition 5. *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation R. For an algorithm $\mathcal{A}$, define experiment* $\mathbf{ZK}_{\mathsf{NIZK}}$ *as follows:*

1. *Compute* $\mathsf{crs}_0 \leftarrow \mathsf{GenZK}(1^\kappa)$ *and* $\mathsf{crs}_1 \leftarrow \mathsf{SimGen}(1^\kappa)$, *and choose a uniform bit $b \leftarrow \{0, 1\}$.*
2. *Run $\mathcal{A}$ on input* crs_b *with access to a* prover oracle PROVE, *which behaves as follows: on input (x, w),* PROVE *returns $\perp$ if $(x, w) \notin R$; otherwise it generates* $\pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}_0, x, w), \pi_1 \leftarrow \mathsf{SimProve}(\mathsf{crs}_1, x, w)$ *and returns π_b.*
3. *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 t,*

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

Definition 6. *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation R. For an algorithm $\mathcal{A}$, define experiment* $\mathbf{SND}_{\mathsf{NIZK}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{GenZK}(1^\kappa)$.
2. *Run $\mathcal{A}$ on input* crs.
3. *When $\mathcal{A}$ outputs (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 t,*

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

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 7 (Simulation Soundness). *Let* $\mathsf{NIZK} = (\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy}, \mathsf{SimGen}, \mathsf{SimProve})$ *be a NIZK for relation* R*. For an algorithm* $\mathcal{A}$*, define experiment* $\mathbf{SIMSND}_{\mathsf{NIZK}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{SimGen}(1^\kappa)$ *and initialize* $\mathcal{Q} := \varnothing$*.*
2. *Run* $\mathcal{A}$ *on input* crs *with access to a* simulated prover oracle SPROVE*, which behaves as follows: on input* (x, w)*,* SPROVE *generates* $\pi \leftarrow \mathsf{SimProve}(x, t)$*, sets* $\mathcal{Q} := \mathcal{Q} \cup \{x\}$*, and returns* π*.*
3. *When* $\mathcal{A}$ *outputs* (x, π)*, the experiment evaluates to* 1 *iff* $x \notin \mathcal{Q}$*,* $\mathsf{Vrfy}(\mathsf{crs}, x, \pi) = 1$*, and* $x \notin \mathcal{L}_R$*.*

NIZK *is* (t, ϵ)-simulation sound *iff for all adversaries* $\mathcal{A}$ *running in time* 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) that 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. (Unfortunately, it does not seem possible to extend this result to prove hardness of the decisional RSW assumption based on factoring in the same model.) 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 [15], 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 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).

In the AGM all algorithms are *algebraic* [8,27]:

Definition 8 (Algebraic Algorithm). *An algorithm $\mathcal{A}$ over $\mathbb{G}$ is called* algebraic *if whenever $\mathcal{A}$ outputs a group element $\mathbf{X} \in \mathbb{G}$, it also outputs an integer vector $\boldsymbol{\lambda}$ with $\mathbf{X} = \prod_i L_i^{\lambda_i}$, where $\boldsymbol{L}$ denotes the (ordered) list of group elements that $\mathcal{A}$ has received as input up to that point.*

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 Sect. 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 performed by an algorithm.

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 is, we require algorithms to be *strongly algebraic*:

Definition 9 (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.*

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] Formally, we require $\mathcal{A}$ to output a flag in its final step to indicate 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 $\kappa + \lceil \log n \rceil$ steps such that the final group element output by $\tilde{\mathcal{A}}$ 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 not an algebraic step, since all computation is "internal" to $\tilde{\mathcal{A}}$ and no group element is being output by $\tilde{\mathcal{A}}$ here.)
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 $\widetilde{\mathsf{Mult}}(\mathbf{X}_1^{\lambda_1}, \ldots, \mathbf{X}_n^{\lambda_n})$.

The theorem follows.

Running Time in the SAGM. The SAGM directly allows us to count the number of algebraic steps used by an algorithm. So far, we have treated all steps in our discussion as algebraic steps. In some settings, however, we may also wish to account for other (non-group) 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 T-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 efficiently factored given any positive multiple of $\phi(N)$ (A proof follows by straightforward adaptation of the proof of [17, Theorem 8.50]):

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 $p', q' \notin \{1, N\}$ such that $N = p'q'$ with probability at least $\frac{1}{2}$.*

We now show:

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

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

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 $\mathcal{R}$ (recursively) 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 $m = 4 \cdot (2^T - x)$. Since, by the claim, $|x| < 2^T$, we have $m \neq 0$ and so m is a nontrivial (integer) multiple of $\phi(N)$ in that case. 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 $t_p + t_o + \theta(\kappa) + 4\lceil \log \alpha \cdot \tau(\kappa) + \tau(\kappa)^2 \rceil$. This completes the proof.

3.3 The RSW Problem in the AGM

In the previous section we have shown 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 at most $t_R \cdot (T+1)$ with* $\Pr\left[\mathbf{FAC}^{\mathcal{M}}_{\mathsf{GenMod}} = 1\right] > \epsilon'$*.*

Proof. Let $\mathcal{R}$ be as described in the theorem statement. Intuitively, $\mathcal{M}$ 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 insofar as analyzing the behavior of $\mathcal{R}$, 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 the answer $(\mathbf{X}, 2^T)$ 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 (i.e., T steps) each time $\mathcal{R}$ calls $\mathcal{A}$.

4 Non-malleable Timed Commitments

In this section 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 *time-released public-key encryption* (TPKE) and show how to construct CCA-secure TPKE.

4.1 Definitions

Timed commitments allow a committer to generate a commitment 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 and provided some specific constructions. We introduce the syntax of *non-interactive* timed commitments and then give appropriate security definitions.

Definition 10. *A* $(t_{cm}, t_{cv}, t_{dv}, t_{fo})$-non-interactive timed commitment scheme (NITC) *is a tuple of algorithms* $\mathsf{TC} = (\mathsf{PGen}, \mathsf{Com}, \mathsf{ComVrfy}, \mathsf{DecomVrfy}, \mathsf{FDecom})$ *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 require that for all $\mathsf{crs} \in \{\mathsf{PGen}(1^\kappa)\}$*, all* $m \in \{0,1\}^\kappa$*, and all* $C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}}$ *output by* $\mathsf{Com}(\mathsf{crs}, m)$*, it holds that*

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

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

To commit to message m, the committer runs Com to get C, π_{Com}, and π_{Decom}, 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_{Decom} to the receiver, who can then run $\mathsf{DecomVrfy}$ to verify the claimed opening. If the committer refuses to decommit, C be opened using FDecom. NITCs are generally only interesting when $t_{fo} \gg t_{cv}, t_{dv}$, i.e., when forced opening of a commitment takes longer than the initial verification and decommitment verification.

NITCs must satisfy appropriate notions of both hiding and binding.

Hiding. For hiding, we introduce a notion of *non-malleability* for NITCs based on the CCA-security notion for (standard) commitments 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, 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 the higher-level protocol) could trick some 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 cost; in contrast, the adversary only incurs a cost of one time unit to make a query to the oracle.

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

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{PGen}(1^\kappa)$*.*
2. *Run* $\mathcal{A}$ *on input* crs *with access to a* decommit oracle $\mathsf{FDecom}(\mathsf{crs}, \cdot)$ *in a preprocessing phase.*
3. *When* $\mathcal{A}$ *outputs* (m_0, m_1)*, choose a uniform bit* $b \leftarrow \{0,1\}$*, compute* $(C, \pi_{\mathsf{Com}}, \star) \leftarrow \mathsf{Com}(\mathsf{crs}, m_b)$*, and run* $\mathcal{A}$ *on input* (C, π_{Com}) *in the online phase.* $\mathcal{A}$ *continues to have access to* $\mathsf{FDecom}(\mathsf{crs}, \cdot)$*, 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* $b' = b$.

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

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

Binding. The binding property states that a commitment cannot be opened to two different messages. It also ensures that the receiver does not accept commitments that cannot be forced open to the correct message.

Definition 12 (BND-CCA Security for Commitments). *For a NITC scheme* TC *and algorithm* $\mathcal{A}$*, define experiment* $\mathbf{BND\text{-}CCA}_{\mathsf{TC}}$ *as follows:*

1. *Compute* $\mathsf{crs} \leftarrow \mathsf{PGen}(1^\kappa)$.
2. *Run $\mathcal{A}$ on input* crs *with access to a* decommit oracle $\mathsf{FDecom}(\mathsf{crs}, \cdot)$.
3. *When $\mathcal{A}$ outputs* $(m, C, \pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}}, m', \pi'_{\mathsf{Decom}})$*, the experiment evaluates to* 1 *iff* $\mathsf{ComVrfy}(\mathsf{crs}, C, \pi_{\mathsf{Com}}) = \mathsf{DecomVrfy}(\mathsf{crs}, C, m, \pi_{\mathsf{Decom}}) = 1$ *and either of the following holds:*
   - $m' \neq m$ *and* $\mathsf{DecomVrfy}(\mathsf{crs}, C, m', \pi'_{\mathsf{Decom}}) = 1$;
   - $\mathsf{FDecom}(\mathsf{crs}, C) \neq m$.

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

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

Time-Released Public-Key Encryption. TPKE can be thought of 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 supports the ability for anyone (and not just the sender) to also recover the message, but only by investing more computational effort.

Definition 13. *A* (t_e, t_{fd}, t_{sd})-timed public-key encryption (TPKE) scheme *is a tuple of algorithms* $\mathsf{TPKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}_f, \mathsf{Dec}_s)$ *with the following behavior:*

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

Such 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 14. *For a TPKE scheme* TPKE *and algorithm $\mathcal{A}$, define experiment* $\textbf{IND-CCA}^{\mathcal{A}}_{\mathsf{TPKE}}$ *as follows:*

1. *Compute $(pk, sk) \leftarrow$* $\mathsf{KGen}(1^\kappa)$*.*
2. *Run $\mathcal{A}$ on input pk with access to a* decryption oracle $\mathsf{Dec}_f(sk, \cdot)$ *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 $\mathcal{A}$ on input 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 $\mathcal{A}$ with preprocessing time t_p and online time t_o,*

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

We remark that in order for TPKE to be an independently interesting primitive, one might require that even for maliciously formed ciphertexts c, Dec_s and Dec_f always produce the same output (a property indeed enjoyed by our TPKE scheme in the next section). However, since our primary motivation is to obtain commitment schemes, we do not require this property and hence opt for a simpler definition that only requires correctness (i.e., of honestly generated ciphertexts).

4.2 CCA-Secure TPKE

Here 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 construction in the ROM in which encryption can be sped up, using 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 contains $\phi(N)$. To encrypt a message $m \in \mathbf{Z}_N$ s.t. $||m|| < \tau(\kappa) - 1$, the sender encodes m as $\mathbf{M} := m^2 \in \mathbb{QR}_N$. It then first computes a random generator $\mathbf{R}$ (by raising g to a random power modulo N), and then 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 the secret key. (To decode to the original m, one can just compute the square root over the integers, since $m^2 < N$).

For any modulus N_1, N_2 and integer T, define the relation

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

Let $(\mathsf{GenZK}, \mathsf{Prove}, \mathsf{Vrfy})$ be a $(t_{pr}, t_v, t_{sgen}, t_{sp})$-NIZK proof system for this relation. Define a TPKE scheme (parameterized by T) as follows:

– $\mathsf{KGen}(1^\kappa)$: For $i = 1, 2$ run $(N_i, p_i, q_i) \leftarrow \mathsf{GenMod}(1^\kappa)$, compute $\phi_i := \phi(N_i) = (p_i - 1)(q_i - 1)$, set $z_i := 2^T \bmod \phi_i$. Choose $g_i \leftarrow \mathbb{QR}_{N_i}$ and run $\mathsf{crs} \leftarrow \mathsf{GenZK}(1^\kappa)$. Output $pk := (\mathsf{crs}, N_1, N_2, g_1, g_2)$ and $sk := (\mathsf{crs}, N_1, N_2, g_1, g_2, z_1, z_2)$.
– $\mathsf{Enc}((\mathsf{crs}, N_1, N_2, g_1, g_2), \mathbf{M})$: For $i = 1, 2$, choose $r_i \leftarrow \mathbb{Z}_{N_i^2}$ and compute

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

where the exponentiations are computed 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 $(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$.
– $\mathsf{Dec}_f((\mathsf{crs}, N_1, N_2, g_1, g_2, z_1, z_2), (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: If $\mathsf{Vrfy}(\mathsf{crs}, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 0$, then output $\perp$. Else compute $\mathbf{Z}_1 := \mathbf{R}_1^{z_1} \bmod N_1$ (using RepSqr) and $\mathbf{M} := \mathbf{C}_1 \mathbf{Z}_1^{-1} \bmod N$, and then output $\mathbf{M}$ if $||\mathbf{M}|| < \tau(\kappa)$ and $\perp$ otherwise.
– $\mathsf{Dec}_s((\mathsf{crs}, N_1, N_2, g_1, g_2), (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi))$: If $\mathsf{Vrfy}(\mathsf{crs}, (\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi) = 0$, then output $\perp$. Else compute $\mathbf{Z}_1 := \mathbf{R}_1^{2^T} \bmod N_1$ (using RepSqr) and $\mathbf{M} := \mathbf{C}_1 \mathbf{Z}_1^{-1} \bmod N_1$, and then output $\mathbf{M}$ if $||\mathbf{M}|| < \tau(\kappa)$ and $\perp$ otherwise..

Fig. 1. A CCA-secure TPKE scheme

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 twice using independent instances of a CPA-secure encryption scheme accompanied by a simulation-sound NIZK proof of consistency between the two ciphertexts. 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 Fig. 1.

Subtleties in the Simulation. The proof of security in our context requires the ability to simulate both the challenge ciphertext and the decryption oracle using a "fast" decryption algorithm. The reason behind this is that if it were not possible to simulate decryption fast, then the reduction from the decisional RSW assumption would take too much time simulating the experiment for the adversary. Fast simulation is possible for two reasons. First, in the proof of the Naor-Yung construction, the simulator knows (at least) one of the secret keys at any time. Second, we use a NIZK with simulation soundness for which verification and proof simulation take linear time in the size of the instance (but not in the size of the circuit). Using these two components, the simulator can perform fast decryption on any correctly formed ciphertext. To reduce from decisional RSW, it embeds the decisional RSW challenge into the challenge ciphertext component for which the secret key is *not* known.

Concretely, for integers N s.t. $N = pq$ for primes p and q, let $\mathcal{C}$ be an arithmetic circuit over $\mathbb{Z}_N$, and let $\mathsf{SAT}_{\mathcal{C}}$ denote the set of all $(x, w) \in \{0,1\}^*$ s.t. w is a satisfying assignment to $\mathcal{C}$ when $\mathcal{C}$'s wires are fixed according to the instance x. The works of Groth and Maller [16] as well as Lipmaa [20] show NIZK constructions for $\mathsf{SAT}_{\mathcal{C}}$ which have soundness and simulation soundness (with suitable parameters), perfect zero-knowledge, perfect correctness and are such that for all $\mathsf{crs} \in \{\mathsf{GenZK}(1^\kappa)\}$, $(\mathsf{crs}', td) \in \{\mathsf{SimGen}(1^\kappa)\}$, all $(x, w) \in \mathsf{SAT}_{\mathcal{C}}$ and all $x' \in \{0,1\}^*$:

- For all $\pi \in \{\mathsf{Prove}(\mathsf{crs}, x, w)\}$, 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 $(\mathsf{crs}', x', \pi')$.
- On input (x', td), $\mathsf{SimProve}$ runs in time $O(|x'|)$.

In other words, both Vrfy and $\mathsf{SimProve}$ run in a fast manner, i.e., linear in the scale of the input instance.

We remark that both of the above constructions work over $\mathbb{Z}_p$ for primes p only, but can be translated to circuits over $\mathbb{Z}_N$, where N is composite, with small overhead, as shown in [18]. The idea is very simple: any arithmetic operation over $\mathbb{Z}_N$ is emulated using multiple (smaller) values in $\mathbb{Z}_p$. The multiplicative overhead in this construction is roughly linear in the size difference between p and N and is ignored here for readability.

Theorem 4. *Suppose* NIZK *is* $(t_p + t_o, 2\epsilon_{ZK})$*-zero-knowledge and* $(t_p + t_o + \theta(\kappa), \epsilon_{SS})$*-simulation sound, and the decisional* T*-RSW problem is* $(t_p + T + t_{sg} + \theta(\kappa), t_o + t_{sp}, \epsilon_{DRSW})$*-hard relative to* GenMod*. Then the* $(t_{pr} + T, t_v + \theta(\kappa), T + \theta(\kappa))$*-TPKE scheme in Fig. 1 is* $(t_p, t_o, \epsilon_{ZK} + \epsilon_{SS} + 2\epsilon_{DRSW})$*-CCA-secure.*

Proof. Let $\mathcal{A}$ be an adversary with preprocessing time t_p and online time t_o. We define a sequence of experiments as follows.

Expt_0: This is the original CCA-security experiment $\textbf{IND-CCA}_{\mathsf{TPKE}}$. 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, in Gen run $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$, 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 $|\Pr[\mathsf{Expt}_1^{\mathcal{A}} = 1] - \Pr[\mathsf{Expt}_0^{\mathcal{A}} = 1]|$ by constructing a reduction $\mathcal{R}_{ZK}$ to the zero-knowledge property of NIZK. $\mathcal{R}_{ZK}$ runs the code of Expt_0, 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^*, for $i = 1, 2$ 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 $\mathcal{R}_{ZK}$ runs $\mathcal{A}(N, g, \mathsf{crs}^*)$.
$\mathcal{R}_{ZK}$ answers $\mathcal{A}$'s DEC queries using the fast decryption algorithm Dec_f. 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 $\mathbf{Z}_1 := \mathsf{RepSqr}(\mathbf{R}_1, N_1, z_1)$ and $\mathbf{M} := \dfrac{\mathbf{C}_1}{\mathbf{Z}_1} \bmod N_1$; 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 for $i = 1, 2$ chooses $r_1, r_2 \leftarrow \mathbb{Z}_{N^2}$, and computes

$$\mathbf{R}_i^* := \mathsf{RepSqr}(g_i, N_i, r_i), \;\; \mathbf{Z}_i^* := \mathsf{RepSqr}(\mathbf{R}_i^*, N_i, z_i), \;\; \mathbf{C}_i^* := \mathbf{Z}_i^* \cdot \mathbf{M} \bmod N_i,$$

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

and outputs $(\mathbf{R}_1^*, \mathbf{R}_2^*, \mathbf{C}_1^*, \mathbf{C}_2^*, \pi^*)$. After that, $\mathcal{R}_{ZK}$ 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.

$\mathcal{R}_{ZK}$ runs in time $t_p + t_o + 2\theta(\kappa)$ (t_p in the setup phase and $t_o + 2\theta(\kappa)$ in the online phase), and

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

Expt_2: Expt_2 is identical to Expt_1, except that $\mathbf{C}_2^*$ is computed as $\mathbf{U}_2 \cdot \mathbf{M}_b \bmod N_2$ (instead of $\mathbf{Z}_2^* \cdot \mathbf{M}_b \bmod N_2$), where $\mathbf{U}_2 := \mathsf{RepSqr}(g_2, N_2, u_2)$ and $u_2 \leftarrow \mathbb{Z}_{N_2^2}$.

We upper 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. $\mathcal{R}_{DRSW}$ runs the code of Expt_2, except that it does not know ϕ_2, and uses the group elements from the decisional T-RSW challenger as part of the challenge ciphertext. (Note that $\mathcal{A}$'s DEC queries can still be answered in a fast manner, since the decryption algorithm only uses $\mathbf{R}_1$, and $\mathcal{R}_{DRSW}$ knows ϕ_1.) Concretely, $\mathcal{R}_{DRSW}$ works as follows:

– Preprocessing phase: $\mathcal{R}_{DRSW}$, on input N, 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 $g_1 \leftarrow \mathbb{QR}_{N_1}$, $g \leftarrow \mathbb{QR}_N$; runs $(\mathsf{crs}, td) \leftarrow \mathsf{SimGen}(1^\kappa)$. Then $\mathcal{R}_{DRSW}$ runs $\mathcal{A}(\mathsf{crs}, N_1, N, g_1, g)$. $\mathcal{R}_{DRSW}$ answers $\mathcal{A}$'s DEC queries as described in Expt_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 $b \leftarrow \{0,1\}$ and $r_1 \leftarrow \mathbb{Z}_{N_1^2}$, and computes

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

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

and returns $(\mathbf{R}_1^*, g^*, \mathbf{C}_1^*, \mathbf{X}^* \cdot \mathbf{M}_b, \pi^*)$. $\mathcal{R}$ answers $\mathcal{A}$'s DEC queries as described in Expt_1.
– Output: On $\mathcal{A}$'s output bit b', $\mathcal{R}_{DRSW}$ outputs 1 if $b' = b$, and 0 otherwise.

$\mathcal{R}_{DRSW}$ runs in time $t_p + t_{sgen}$ in the preprocessing phase, and time $t_o + t_{sprove}$ in the online phase, and

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

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 conceptual change, so

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

Expt_4: Expt_4 is identical to Expt_3, except that the DEC oracle uses $\mathbf{R}_2$ (instead of $\mathbf{R}_1$) to decrypt. That is, when $\mathcal{A}$ queries $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, compute $\mathbf{Z}_2 := \mathsf{RepSqr}(\mathbf{R}_2, N_2, z_2)$ and $\mathbf{M} := \dfrac{\mathbf{C}_2}{\mathbf{Z}_2} \bmod N_2$.

Expt_4 and Expt_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. $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1 \neq \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$ (over $\mathbb{Z}$) but $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 1$ (in which case $\mathcal{A}$ receives $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1$ in Expt_3 and $\dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$ in Expt_4; in all other cases $\mathcal{A}$ receives either $\perp$ in both experiments, or $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1 = \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$ in both experiments). Denote this event Fake. We upper bound $\Pr[\mathsf{Fake}]$ by constructing a reduction $\mathcal{R}_{SS}$ to the simulation soundness of NIZK:

– Setup: $\mathcal{R}_{SS}$, on input crs, for $i = 1, 2$ 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 $\mathcal{A}(N, g, \mathsf{crs})$.

  On $\mathcal{A}$'s query $\mathsf{DEC}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi)$, $\mathcal{R}_{SS}$ computes $\mathbf{Z}_1$ and $\mathbf{Z}_2$ as described in Expt_1. 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 $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1 = \dfrac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$, and if so, it returns $\dfrac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1$, 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

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

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

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

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

$\mathcal{R}_{SS}$ runs in time at most $t_p+t_o+\theta(\kappa)$ (i.e., t_p in the setup phase and $t_o+\theta(\kappa)$ in the online phase). Up to the point that $\mathcal{R}_{SS}$ outputs, $\mathcal{R}_{SS}$ simulates Expt_4 perfectly. If Fake happens, then $\mathcal{R}_{SS}$ outputs $((\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2), \pi)$ s.t. $\frac{\mathbf{C}_1}{\mathbf{R}_1^{2^T}} \bmod N_1 \neq \frac{\mathbf{C}_2}{\mathbf{R}_2^{2^T}} \bmod N_2$ but $\mathsf{Vrfy}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{C}_1, \mathbf{C}_2, \pi) = 1$, winning the simulation-soundness experiment. It follows that

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

Expt_5: Expt_5 is identical to Expt_4, except that $\mathbf{C}_1^*$ is computed as $\mathbf{U} \cdot \mathbf{M}_b \bmod N_1$ (instead of $\mathbf{Z}_1^* \cdot \mathbf{M}_b \bmod N_1$), where $\mathbf{U}_1 := \mathsf{RepSqr}(g_1, N_1, u_1)$ and $u_1 \leftarrow \mathbb{Z}_{N_1^2}$. The argument is symmetric to the one from Expt_1 to Expt_2; the reduction works because $\mathbf{R}_1$ is not used in DEC. We have

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

Expt_6: Expt_6 is identical to Expt_5, except that $\mathbf{C}_1^*$ is computed as $\mathbf{U}_1$ (instead of $\mathbf{U}_1 \cdot \mathbf{M}_b$). The argument is symmetric to the one from Expt_2 to Expt_3. We have

$$\Pr[\mathsf{Expt}_6^{\mathcal{A}} = 1] = \Pr[\mathsf{Expt}_5^{\mathcal{A}} = 1].$$

Furthermore, since b is independent of $\mathcal{A}$'s view in Expt_6, we have

$$\Pr[\mathsf{Expt}_6^{\mathcal{A}} = 1] = \frac{1}{2}.$$

Summing up the results above, we conclude that

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

which completes the proof.

4.3 Constructing Non-malleable Timed Commitments

In this section, we show how our notion of CCA-secure TPKE implies non-malleable timed commitments. The idea is very simple. At setup, the committer generates the parameters and keys for a TPKE TPKE and NIZKs $\mathsf{NIZK}_{\mathsf{Com}}$ and $\mathsf{NIZK}_{\mathsf{Decom}}$. To commit to a message m, the committer computes $c := \mathsf{Enc}(pk, m; r)$ (for some random coins r) and uses $\mathsf{NIZK}_{\mathsf{Com}}$ and $\mathsf{NIZK}_{\mathsf{Decom}}$ to prove that (1) it knows (m, r) s.t. $c = \mathsf{Enc}(pk, m; r)$. This proof will be used as π_{Com}, i.e., to prove that the commitment is well-formed; and (2) it knows r s.t. $c = \mathsf{Enc}(pk, m; r)$. This proof will be used as π_{Decom}, i.e., to prove (efficiently) that the opening to the commitment is the correct one. Our construction is presented in Fig. 2.

To be able to reduce from CCA-security of the underlying TPKE scheme for meaningful parameters, we require that proofs of the NIZK scheme can be simulated and verified (very) efficiently, i.e., take much less time than a forced decommit. This is satisfied when instantiating the TPKE scheme with our construction from the previous section, where this relation can be expressed via an arithmetic circuit. More generally, any scheme whose encryption algorithm can be expressed via an arithmetic circuit would satisfy our requirements.

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

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

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

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

Define an NITC scheme as follows:

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

Fig. 2. An NITC scheme.

Correctness of this scheme follows immediately from correctness of the underlying TPKE and NIZK schemes; we next show its CCA-security.

Theorem 5. *Suppose* TPKE *is* $(t_p + t_{csgen}, t_{csp}, \epsilon_{TPKE})$*-CCA-secure, and* $\mathsf{NIZK}_{\mathsf{Com}}$ *is* $(t_p + t_o + t_e, \epsilon_{ZK})$*-zero-knowledge. Then the* $(t_e + \max\{t_{cp}, t_{dp}\}, t_{cv}, t_{dv}, t_{sd})$*-NITCS scheme in Fig. 2 is* $(t_p, t_o, \epsilon_{ZK} + \epsilon_{CCA})$*-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 $\mathsf{crs}_{\mathsf{Com}}$ and π^* are simulated. That is, in the setup phase run $(\mathsf{crs}_{\mathsf{Com}}, td) \leftarrow \mathsf{SimGen}_{\mathsf{Com}}(1^\kappa)$, and in the challenge compute $\pi^* \leftarrow \mathsf{SimProve}_{\mathsf{Com}}(c^*, td)$.

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

– Setup: $\mathcal{R}_{ZK}$, on input crs^*, runs $P \leftarrow \mathsf{PGen}(1^\kappa)$, $(sk, pk) \leftarrow \mathsf{KGen}(P)$ and $\mathsf{crs}_{\mathsf{Decom}} \leftarrow \mathsf{GenZK}_{\mathsf{Decom}}(1^\kappa)$, sets $\mathsf{crs} := (pk, \mathsf{crs}^*, \mathsf{crs}_{\mathsf{Decom}})$, and runs $\mathcal{A}(\mathsf{crs})$. 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.

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

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

Now 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, and DEC simply runs Dec_s, $\mathcal{A}$'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 $(\mathsf{crs}_{\mathsf{Com}}, td) \leftarrow \mathsf{SimGen}_{\mathsf{Com}}(1^\kappa)$, and runs $\mathcal{A}(\mathsf{crs}_{\mathsf{Com}})$. On $\mathcal{A}$'s query $\mathsf{DEC}(c)$, $\mathcal{R}_{CCA}$ queries $\mathsf{DEC}_{\mathsf{TPKE}}(c)$ 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.
– Output: When $\mathcal{A}$ outputs a bit b', $\mathcal{R}_{CCA}$ also outputs b'.

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

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

Summing up all results above, we conclude that

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

which completes the proof.

We give a sketc.h of the argument of why our scheme satisfies our notion of binding. Recall that if $\mathcal{A}$ can win $\textbf{BND-CCA}_{\mathsf{TC}}$, then it can produce a commitment c along with messages m, m' and proofs $\pi_{\mathsf{Com}}, \pi_{\mathsf{Decom}}$ s.t. $\mathsf{ComVrfy}((pk, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{Decom}}), c, \pi_{\mathsf{Com}}) = \mathsf{DecomVrfy}((pk, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{Decom}}), c, m, \pi_{\mathsf{Decom}}) = 1$, $m' \neq m$ and either

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

or

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

Both (1) and (2) can be reduced from soundness of NIZK. For (1), unless $\mathcal{A}$ can come up with a fake proof π_{Com}, then $\mathsf{ComVrfy}((pk, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{Decom}}), c, \pi_{\mathsf{Com}}) = 1$ implies that there exists m and r s.t. $\mathsf{Enc}(pk, m; r) = c$. Now, correctness of TPKE implies that $\mathsf{FDecom}((pk, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{Decom}}), c) = \mathsf{Dec}_s(pk, c) = \mathsf{Dec}_f(sk, c) = m$. Similarly, for (2), unless $\mathcal{A}$ can come up with a fake proof π_{Decom}, then $\mathsf{DecomVrfy}((pk, \mathsf{crs}_{\mathsf{Com}}, \mathsf{crs}_{\mathsf{Decom}}), c, m, \pi_{\mathsf{Decom}}) = 1$ implies that there exists r s.t. $\mathsf{Enc}(pk, m; r) = c$. In this case, correctness of TPKE asserts that $\mathsf{Dec}_s(pk, c) = \mathsf{Dec}_f(sk, c) = m \neq m'$. Hence the proof π'_{Decom} must be fake, as otherwise, this would contradict correctness of TPKE with regard to m'.

References

1. Aggarwal, D., Maurer, U.: Breaking RSA generically is equivalent to factoring. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 36–53. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_2
2. Baum, C., David, B., Dowsley, R., Nielsen, J.B., Oechsner, S.: Tardis: time and relative delays in simulation. Cryptology ePrint Archive: report 2020/537 (2020)
3. Bellare, M., Goldwasser, S.: Verifiable partial key escrow. In: ACM Conference on Computer and Communications Security (CCS) 1997, pp. 78–91. ACM Press (1997)
4. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: Sudan, M., (ed.) ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science, pp. 345–356, Cambridge, MA, USA, 14–16 Jan 2016. Association for Computing Machinery (2016)
5. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology-Crypto 2018, Part I. LNCS, vol. 10991, pp. 757–788. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96884-1_25
6. Boneh, D., Bünz, B., Fisch, B.: A survey of two verifiable delay functions. Cryptology ePrint Archive, report 2018/712 (2018). https://eprint.iacr.org/2018/712
7. Boneh, Dan, Naor, Moni: Timed commitments. In: Bellare, Mihir (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 236–254. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_15
8. Boneh, D., Venkatesan, R.: Breaking RSA may not be equivalent to factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 59–71. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054117

9. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science (FOCS), pp. 136–145. IEEE (2001)
10. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: 51st Annual Symposium on Foundations of Computer Science (FOCS), pp. 541–550. IEEE (2010)
11. Cathalo, J., Libert, B., Quisquater, J.-J.: Efficient and non-interactive timed-release encryption. In: Qing, S., Mao, W., López, J., Wang, G. (eds.) International Conference on Information and Communication Security (ICICS). LNCS, vol. 3783, pp. 291–303. Springer, Heidelberg (2005). https://doi.org/10.1007/11602897_25
12. Di Crescenzo, G., Ostrovsky, R., Rajagopalan, S.: Conditional oblivious transfer and timed-release encryption. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 74–89. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_6
13. Dwork, C., Naor, M.: Zaps and their applications. In: 41st Annual Symposium on Foundations of Computer Science (FOCS), pp. 283–293. IEEE (2000)
14. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive: report 2020/779 (2020)
15. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology-Crypto 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
16. Groth, Jens, Maller, Mary: Snarky signatures: minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, Jonathan, Shacham, Hovav (eds.) CRYPTO 2017. LNCS, vol. 10402, pp. 581–612. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63715-0_20
17. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, 2nd edn. Chapman & Hall/CRC Press, CRC Press, Boca Raton, London, New York (2014)
18. Kosba, A., et al.: How to use SNARKs in universally composable protocols. Cryptology ePrint Archive, report 2015/1093 (2015). http://eprint.iacr.org/2015/1093
19. Lin, H., Pass, R., Soni, P.: Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In: 58th Annual Symposium on Foundations of Computer Science (FOCS), pp. 576–587. IEEE (2017)
20. Lipmaa, H.: Simulation-extractable SNARKs revisited. Cryptology ePrint Archive, report 2019/612 (2019). https://eprint.iacr.org/2019/612
21. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Crypt. **86**(11), 2549–2586 (2018). https://doi.org/10.1007/s10623-018-0461-x
22. Mahmoody, Mohammad, Moran, Tal, Vadhan, Salil: Time-lock puzzles in the random oracle model. In: Rogaway, Phillip (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 39–50. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_3
23. Malavolta, Giulio, Thyagarajan, Sri Aravinda Krishnan: Homomorphic time-lock puzzles and applications. In: Boldyreva, Alexandra, Micciancio, Daniele (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 620–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_22
24. Maurer, Ueli: Abstract models of computation in cryptography. In: Smart, Nigel P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (2005). https://doi.org/10.1007/11586821_1
25. May, T.: Timed-release crypto (1993). http://cypherpunks.venona.com/date/1993/02/msg00129.html

26. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen Ciphertext attacks. In: 22nd Annual ACM Symposium on Theory of Computing (STOC), pp. 427–437. ACM Press (1990)
27. Paillier, Pascal, Vergnaud, Damien: Discrete-log-based signatures may not be equivalent to discrete log. In: Roy, Bimal (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_1
28. Pietrzak, K.: Simple verifiable delay functions. In: Blum, A. (eds.) ITCS 2019: 10th Innovations in Theoretical Computer Science Conference, vol. 124, pp. 60:1–60:15, San Diego, CA, USA, 10–12 Jan 2019. Leibniz International Proceedings in Informatics (LIPIcs)
29. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical report, MIT Laboratory for Computer Science (1996)
30. Rotem, Lior, Segev, Gil: Generically speeding-up repeated squaring is equivalent to factoring: sharp thresholds for all generic-ring delay functions. In: Micciancio, Daniele, Ristenpart, Thomas (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 481–509. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_17
31. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th Annual Symposium on Foundations of Computer Science (FOCS), pp. 543–553. IEEE (1999)
32. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
33. Wesolowski, B.: Efficient verifiable delay functions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 379–407. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_13