Timed Commitments Revisited

Miguel Ambrona, Marc Beunardeau, and Raphaël R. Toledo

Nomadic Labs, Paris, France
name.surname@nomadic-labs.com

Abstract. Timed commitments (Boneh and Naor, CRYPTO 2000) are a variant of standard commitments which incorporates a forced opening mechanism that allows anyone to reveal the committed message, but not before a certain prescribed date. Timed commitments have a wide-range of applications such as contract signing, fair multi-party computation, sealed bid auctions or new blockchain applications such as preventing front-running or unbiased randomness generation.
We revisit the notion of timed commitments and propose an alternative simplified definition. We also provide two new constructions of timed commitments with different trade-offs.

1 Introduction

Timelock puzzles were conceived by Rivest, Shamir and Wagner [RSW96] as a tool for "sending information into the future". They allow a sender to encrypt a message that can eventually be decrypted, but not before a certain (approximated) future date chosen by the sender.

Concretely, given a function $F : \mathcal{X} \to \mathcal{Y}$, a timelock puzzle is defined by an input $x \in \mathcal{X}$ and the puzzle solution is the output $y := F(x)$. It is required that evaluating F involve a computation that is inherently sequential (not known to be parallelizable). This guarantees that evaluating F, i.e. "solving puzzles", cannot be speeded up by leveraging extra computational power.[1]

The iterated application of a secure hash function would be a good candidate function F that cannot be parallelized. However, that would require the puzzle creator to invest a significant amount of computational effort to arrive to the solution. Indeed, the same amount as a third party solving the puzzle. Ideally, it should be possible for the puzzle creator to generate a valid puzzle-solution pair efficiently. Rivest et al. show that this is possible [RSW96] if F is implemented as an iterated squaring over a group of hidden order:

$$\begin{aligned} \textit{puzzle:} \quad & x := (G, g, T), \text{ where } G \text{ is a group, } g \in G \text{ and } T \in \mathbb{N} \\ \textit{solution:} \quad & y := g^{2^T} \end{aligned}$$

The above computation is conjectured to require T iterated squarings for a party who does not know the group order (if the group and its representation are chosen appropriately). On the other hand, knowing the group order η can dramatically speed up the evaluation: simply power g to $(2^T \bmod \psi)$. Consequently, η must be hidden from the public, but if it is known to the puzzle creator, they compute the solution very efficiently. Rivest et al. achieve this by considering an RSA group, where the modulus is chosen by the puzzle creator.

Sending information into the future. As mentioned earlier, timelock puzzles can be used to commit to a piece of information that will eventually be revealed, after a certain (controllable) amount of time has passed. For that, the "sender" can create a puzzle-solution pair (x, y), derive a secret key from y and use it to encrypt the message of their choice, publishing puzzle x together with the corresponding ciphertext. This would allow a third party to compute y by solving the puzzle, deriving the key from it, and decrypting the ciphertext, recovering the original message.

Rivest et al. proposed several other applications of timelock puzzles, including fair auctions, scheduled payments, or delayed key-escrow schemes. Remarkably, timelock puzzles can also be used to address a recent but important problem, known as *generalized front-running*, which affects many permissionless blockchains (see Section 1.2).

[1] Although, it can unavoidably be speeded up by utilizing a faster machine, or through hardware acceleration, e.g. using FPGA or ASIC.

Verifiable delay functions. A verifiable delay function (VDF) [BBBF18] is a function $F : \mathcal{X} \to \mathcal{Y}$ that requires a prescribed amount of time to be evaluated, even when using parallelization. In this sense, it is similar to the underlying function of a timelock puzzle. However, the V in VDF requires that the validity of a claimed output y can be efficiently verified by anyone. On the other hand, unlike timelock puzzles, VDFs do not require that the computation of y be efficient for the party who chooses x. Indeed, they require that the computation be inefficient for everyone (note that VDFs are more suited in applications where the actors have a more symmetric role in the protocol, contrary to timelock). We refer to [Wes19, Pie19] for more details about VDF. For the purposes of this paper, we will describe Pietrzak's VDF argument (see Section 2.4).

Verifiable delay functions have shown to be useful for enforcing delays in the area of decentralized applications, where efficient public verifiability is paramount. In particular, they can be used for leader election, proofs of replication, and randomness generation (see Section 1.2). Indeed, VDFs have been adopted by several popular blockchains such as Tezos or Chia as part of their protocols, and other chains such as Ethereum or Filecoin are working on their integration.

Timed commitments. Timed commitments were introduced by Boneh and Naor [BN00] as a variant of standard commitments that allows anyone to reveal the committed message through a forced opening mechanism. However, such mechanism is designed to be computationally expensive and non-parallelizable to ensure that the message will remain hidden until an approximated future date. Furthermore, once a commitment is opened, anyone can efficiently verify the correctness of the opened message. More concretely, timed commitments were originally defined to be interactive protocols with a committing phase and a (forced) opening phase and they were required to achieved the following properties:

(i) *Guaranteed opening*: after the commit phase, the receiver gets convinced that the forced opening will be successful (before actually performing the computation).
(ii) *Provable opening*: the (forced) opening phase results in a certificate of the validity of the opened value.
(iii) *Sequentiality*: the forced opening phase cannot be significantly speeded up with parallelization.

Timed commitments can thus be seen as a primitives that combine properties from both timelock puzzles (fast generation and sequentiality) and from verifiable delay functions (verifiability).

Since the first construction by Boneh and Naor, other works have explored the notion of timed commitments (especially in the non-interactive setting) and have considered additional security properties such as *non-malleability* [KLX20]. However, the constructions in [KLX20] make use of general-purpose zero-knowledge proofs and their commitment phase is very inefficient, requiring an expensive computation by the committing party. Other works [TCLM21] study timed commitments over class groups, to avoid trusted setups, and focus on commitment protocols involving multiple parties. Note that class groups are very new primitives whose security needs to be more consolidated and studied further. Other remarkable works are [MA22, CJ22], but they also rely on zero-knowledge proofs.

In this work, we aim at constructing timed commitment schemes whose efficiency is suitable for practical applications. We thus focus on constructions with no trusted setup and that do not rely on the use of zero-knowledge proofs.

1.1 Our contributions

We revisit and simplify the notion of timed commitments. In particular, we remove the condition of *guaranteed opening*. Namely, we do not require that there exist a mechanism to guarantee that the forced opening of the commitment will be successful. Instead, we consider a special message $\perp$, which is the default opening value for commitments whose forced opening is unsuccessful. Furthermore, our mechanisms for verifiability of openings make it possible to prove that a commitment is "invalid" (that it forced opens to $\perp$). One could argue that this simplified definition leads to a primitive that is strictly weaker, as it is no longer possible to know in advance whether the forced opening will be successful. We claim that this is not the case, as this problem is inherent: under the original definition of timed commitments a malicious party can create a valid commitment to a meaningless message (there is no way to know in advance whether the result of forced

opening will be valid in the language induced by the use case, e.g., that it will be a well-formed blockchain transaction).

We establish formal security definitions (Section 3) for our new presentation that facilitate a simple and rigorous security analysis when used for applications such as preventing the problem of block producer extractable value, and efficient, unbiased randomness generation (see Section 1.2). We also provide two constructions of timed commitments with different trade-offs and formally prove that they meet our security requirements (Section 4).

1.2 Applications

Preventing generalized front-running. The problem of generalized front-running, also referred to as as *block producer extractable value (BPEV)* in proof-of-stake chains or as *miner extractable value (MEV)* in proof-of-work blockchains, is a common attack that affects permissionless blockchains where transactions can be observed before they are actually included on-chain. This allows block producers to benefit from the fact that they can choose the order of transactions within a block. For example, upon receiving a transaction, a block producer can craft a block including this transaction and one of their own in such a way that the sequential execution of both transactions guarantees a profit.

This problem can be partially mitigated by leveraging commitment schemes: users commit to their transactions, which are included on-chain in committed form and whose execution is postponed until they are revealed. Unfortunately, this solution allows malicious users to open their commitments selectively depending on their own interest.

Using timed commitments instead would solve this issue. There could be a party (possibly a block producer) who is prepared to open commitments of users who denied to collaborate[2], and produce proofs of correctness of these openings, which can be efficiently verified on-chain.

Randomness generation. Generation of uniform and unbiased randomness is a critical problem in decentralized systems. The RANDAO protocol is a widely used approach to solve this problem and, in a nutshell, works as follows. Every party participating in the process of randomness generation samples a string (of a prescribed length) uniformly at random and commits to it, transmitting the commitment to the other parties. When all parties have received all commitments, they start the revelation phase, where each opens their own commitment. The output of the protocol is then computed by aggregating all such strings, e.g., by XORing them, which guarantees that as long as one of the users chose their string uniformly at random, the final output is uniformly distributed.

Unfortunately, this simple protocol is not guaranteed to terminate. What if a party does not open their commitment in the revelation phase? A natural protocol modification to guarantee termination would be to exclude that party's value in the computation of the final output. However, that allows a malicious party to bias the result. Namely, they can wait until everybody has opened their commitment and then decide whether or not they want open their own (being able to choose between two possible outcomes). This problem worsens when n parties collude, being able to choose between 2^n possible outcomes.

In order to mitigate the above problem, some important blockchains such as Tezos or Chia have started to use verifiable delay functions as part of their randomness generation protocols. For example, by using the output of RANDAO as the input of a VDF function. If the VDF difficulty is properly adjusted, the malicious parties trying to bias the output of RANDAO would not have enough time to evaluate the VDF during the revelation phase. Therefore, the decision of whether or not they open their commitment cannot be based on what the final outcome will be. Furthermore, the public verifiability of VDFs makes it possible to produce a certificate of the fact that the result of the protocol is correct. This allows the blockchain nodes to avoid the expensive VDF evaluation and simply verify such certificate.

Nevertheless, VDFs are very new primitives which deserve more study. The VDF difficulty should probably be overestimated in order to account for possible hardware acceleration. This makes VDF solutions for randomness generation significantly more expensive than RANDAO (not for the blockchain nodes, but for

[2] Such users may be penalized in order to discourage such behavior.

the party who evaluates the VDF), affecting the protocol duration. Replacing standard commitments with timed commitments in the RANDAO protocol could mitigate this issue. If the scheme has the property that commitment creators can produce proofs of opening of their own commitments very efficiently, and they collaborate to do so, we could avoid having to bruteforce any opening: leading to a very efficient protocol in the optimistic case that parties are honest. Dishonest behavior can be discouraged through economic incentives and will not lead to any biases in randomness generation since the commitments of users who do not collaborate will eventually be opened too.

1.3 Technical overview

The underlying function used by the timelock puzzle scheme by Rivest et al. [RSW96], i.e. iterated squaring over a group, is the same function considered by the two most popular VDFs [Wes19, Pie19]. Therefore, one could be tempted to build a timed commitment by equipping the timelock puzzle by Rivest et al. with the verification mechanisms devised by Wesolowski or Pietrzak. However, this attempt does not work because knowing the group order compromises the soundness of both VDF arguments. Indeed, with the group order, an adversary could break the *adaptive root assumption* and the *low order assumption*, the two security assumptions on which Wesolowski's argument and Pietrzak's argument respectively rely. Under these conditions, the commitment creator would have the ability to fake opening proofs.

Remark 1. If we used *authenticated encryption* to encrypt the committed message, this would ensure that a commitment can only be opened to one message. However, this would not counter the fact that the commitment creator can forge puzzle-solution proofs, because we need to account for invalid commitments. Indeed, the following attack would be possible. Let x be the timelock puzzle chosen by the creator and let y be its solution. The creator could select a private-key $k \neq y$ and encrypt the desired message m with it, producing ct. The pair (x, ct) would be an *invalid* commitment, as it is not well-formed: a party bruteforcing the puzzle would arrive to y and realize that decryption of ct with y fails. However, the puzzle creator could forge a VDF proof of the (false) fact that (x, k) is a valid puzzle-solution pair, thus convincing anyone that m is the correct opening of the commitment.

Note that for the use cases that we devised in Section 1.2, our new definition of timed commitments must handle invalid commitments: it should be possible to prove/verify that a commitment is malformed (and does not force open to any message). As we have seen, a commitment creator who has the ability to forge puzzle-solution proofs can produce a commitment of a message and then prove that it opens to such message, but they can also prove that the commitment is invalid. The timed commitment definition of soundness should prevent such situation. On the other hand, it is not clear how to achieve quick commitment generation, while hiding the group order from all parties (including the commitment creator).

Interestingly, the VDF argument by Pietrzak could be sound even against an adversary who knows the group order, if the group does not contain low order elements: in that case the low order assumption would hold unconditionally. (Pietrzak proposed a mechanism to build groups of hidden order with this property[3], see Figure 2). However, the previous attempt of timed commitments falls short even when implemented over such *rough* groups and using Pietrzak's argument. The reason is that the commitment creator could deviate from the scheme and choose a group that is not rough. This cannot be noticed by a third party (at least with Pietrzak's implementation of rough groups) and, again, would give the creator the ability to forge proofs of opening.

The previous issue could be overcomed by including zero-knowledge proofs that ensure that the group is rough, but this would incur an undesirable overhead. Instead, our first construction resolves the problem by embedding some extra piece of information in the committed message from which anyone can verify the

[3] This group implementation requires sampling safe primes, i.e., primes of the form $2p+1$ where p is also prime. This can be considered costly, but our experiments show that it is possible to generate 1024-bit safe primes in under a second (on average) with OpenSSL running on a personal laptop 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz. This bit length should be enough for most blockchain applications.

roughness of the group (see Section 4.1). This creates a loop that prevents the previous attack: if the creator selects a group that is not rough, they can fake puzzle-solution proofs, but they will not be able to be able to convince anyone that a commitment is valid, as a witness of the roughness of the group does not exist.

We propose a second construction that works with both Pietrzak's and Wesolowski's arguments and any algebraic group where such arguments are sound. (In this case, the group order must be hidden from all parties, including the commitment creator). This second construction is based on the idea that a puzzle-solution pair can be rerandomized. In particular, if (x, y) is a valid puzzle-solution pair to the iterated squaring function, so is the pair (x^r, y^r), for any $r \in \mathbb{N}$. This way, at the cost of introducing a setup phase where a valid puzzle-solution pair is computed for the desired difficulty T, we can then generate arbitrarily many puzzle-solution pairs very efficiently, without the group order. Unfortunately, this does not seem to be enough to achieve a desirable property that we mentioned earlier: it should be possible for the commitment creator to produce a proof of opening efficiently. We refer to Section 4.2 for details on how we achieve this extra property by integrating the previous idea in a slightly different manner.

2 Preliminaries

2.1 Notation

For $n \in \mathbb{N}$, we define $\mathbb{Z}_n \coloneqq \mathbb{Z}/n\mathbb{Z}$, the ring of integers modulo n. We denote its group of units by $\mathbb{Z}_n^*$. If $a, b \in N$ are congruent modulo n, we write $a =_n b$. We denote the Jacobi symbol of a modulo n by $\left(\frac{a}{n}\right)$. We use multiplicative notation for abstract algebraic groups. A safe prime is a prime number of the form $2p+1$ where p is also prime. Given two functions $f, g : \mathbb{N} \to [0, 1]$, we write $f \approx g$ if the difference $|f(\lambda) - g(\lambda)|$ is asymptotically smaller than the inverse of any polynomial. Function f is said to be *negligible* if $f \approx 0$, whereas it is said to be *overwhelming* when $f \approx 1$. We denote by $\{0,1\}^*$ the set of strings of arbitrary length. We define a special symbol for concatenation $\|$ that allows for unambiguous parsing. Namely, $(m_1 \,\|\, m_2) = (m_1' \,\|\, m_2')$ implies that $m_1 = m_1'$ and $m_2 = m_2'$, for all $m_1, m_2, m_1', m_2' \in \{0,1\}^*$.

2.2 Hidden order groups

Let $\mathcal{G}$ be an algorithm that on input a security parameter in unary 1^λ, outputs the description of a group G of unknown order, together with a trapdoor τ. The group description allows to efficiently compute the group law, test membership and perform uniform (or statistically close to uniform) sampling over G. Furthermore, using trapdoor τ, it is possible to derive the group's order and checking λ-roughness of the group's order. In particular, we assume the existence of algorithms order and $\lambda\text{-}\mathsf{rough}$ taking (G, τ) as input and returning an integer and a bit respectively. Furthermore, we require that, for all $\lambda \in \mathbb{N}$, $\Pr\left[(G, \tau) \leftarrow \mathcal{G}(1^\lambda) : \lambda\text{-}\mathsf{rough}(G, \tau) = 1\right] = 1$ and that for every G, $\exists \tau^*$, $\lambda\text{-}\mathsf{rough}(G, \tau^*) = 1$ imply that G is well-defined, and that $\mathsf{order}(G, \tau^*)$ is the correct order of G, which is λ-rough (all its prime divisors are greater than 2^λ).

2.3 Encryption

We require an encryption scheme to allow decryption in the future with the weak security property of indistiguishability in the presence of an eavesdropper.

Definition 1. *A* private-key encryption *scheme is a triple of PPT algorithms:*

- $\mathsf{Gen}(1^\lambda) \to k$*, on input a security parameter, outputs a key.*
- $\mathsf{Enc}_k(m) \to \mathsf{ct}$*, on input a key* k *and a message* $m \in \{0,1\}^*$*, outputs a ciphertext.*
- $\mathsf{Dec}_k(\mathsf{ct}) \to m/\perp$*, on input a key* k *and a ciphertext* ct*, (deterministically) outputs a message or* $\perp$*.*

A private-key encryption scheme is *correct* if for every m,

$$\Pr[k \leftarrow \mathsf{Gen}(1^\lambda) : m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))] = 1 \ .$$

Let $\mathsf{H} : \{0,1\}^* \to \{1,\ldots,2^\lambda\}$ be a hash function (modeled as a random oracle).

Pietrzak.Prove(G, T, g, h):

Inputs: group description G, time-bound $T = 2^t \in \mathbb{N}$, group elements $g, h \in G$

Output: proof ensuring that $g^{2^T} = h$

1: let $g_0 \coloneqq g$ and $h_0 \coloneqq h$

2: **for every** $i = 1, \ldots, t$ **do**:

3: compute $v_i \coloneqq g^{2^{(T/2^i)}}$, sample $r \coloneqq \mathsf{H}(v_i \,\|\, g_{i-1} \,\|\, h_{i-1} \,\|\, i)$[a], and set $g_i \coloneqq g_{i-1}^r v_i$ and $h_i \coloneqq v_i^r h_{i-1}$

4: **return** $\pi \coloneqq \{v_i, g_i, h_i\}_{i\in[t]}$

Pietrzak.Verify(G, T, g, h, π):

Inputs: group description G, time-bound $T = 2^t \in \mathbb{N}$, group elements $g, h \in G$ proof π

Output: 1/0 (indicating acceptance or rejection, respectively)

1: **if** $g \notin G$ or $h \notin G$ **then return** 0

2: parse π as $\{v_i, g_i, h_i\}_{i\in[t]}$, let $g_0 \coloneqq g$ and $h_0 \coloneqq h$, and compute $r_i \coloneqq \mathsf{H}(v_i \,\|\, g_{i-1} \,\|\, h_{i-1} \,\|\, i)\ \forall i \in [t]$

3: **if** $\exists i \in [t] : v_i \notin G$ or $g_i \notin G$ or $h_i \notin G$ or $g_i \neq g_{i-1}^{r_i} v_i$ or $h_i \neq v_i^{r_i} h_{i-1}$ **then return** 0

4: **return** $(h_t = g_t^2)$

Fig. 1: Pietrzak's succinct argument for relation $R(G, T, g, h) \coloneqq g, h \in G \ \wedge\ g^{2^T} = h$.

[a] Including i in the hash is not necessary for soundness, but it leads to a better bound, independent of T.

Definition 2 (Indistiguishability in the presence of an eavesdropper). *A private-key encryption scheme is indistiguishability in the presence of an eavesdropper if for every (stateful) PPT algorithm $\mathcal{A}$, the following probability is negligibly close to $\frac{1}{2}$ in λ:*

$$\Pr\left[(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda);\ k \leftarrow \mathsf{Gen}(1^\lambda);\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{Enc}_k(m_b);\ b' \leftarrow \mathcal{A}(\mathsf{ct})\ :\ b = b'\right] .$$

2.4 Pietrzak's VDF argument

As a component of his VDF, Pietrzak proposed a public-coin interactive argument for relation $R(G, T, g, h) \coloneqq g, h \in G \wedge g^{2^T} = h$ [Pie19]. The protocol can be proven secure under the low order assumption, which states that it is computationally hard to find non-trivial elements of low order in a given group:

Definition 3 (Low order assumption). *The low order assumption holds for algorithm $\mathcal{G}$ if for PPT adversary $\mathcal{A}$ the following probability is negligible in λ:*

$$\Pr\left[(G, \_) \leftarrow \mathcal{G}(1^\lambda);\ (\mu, d) \leftarrow \mathcal{A}(G)\ :\ \mu \in G \ \wedge\ \mu \neq 1_G \ \wedge\ 1 < d < 2^\lambda \ \wedge\ \mu^d = 1_G\right] ,$$

where the probability is taken over the coins of $\mathcal{G}$ and $\mathcal{A}$.

If the group order is rough (it does not contain "small" prime factors), by virtue of Lagrange's theorem, the low order assumption holds unconditionally, even if the adversary is allowed to choose the group or knows the group order. Remarkably, in such groups Pietrzak's argument achieves statistical soundness.

$\mathcal{G}(1^\lambda)$: sample $\ell(\lambda)$-bits safe primes p, q; set $N = p \cdot q$; **return** $(G \coloneqq N, \tau \coloneqq p)$

$x \in G$: **return** $1 \leq x \leq (N-1)/2 \wedge \left(\frac{x}{N}\right) = +1$

$x \circ y$: **return** $|x \cdot y \bmod N|$

$\mathsf{order}(G \coloneqq N, \tau \coloneqq p)$: let $q = N/p$; **return** $\frac{(p-1)(q-1)}{4}$

$\lambda\text{-}\mathsf{rough}(G \coloneqq N, \tau \coloneqq p)$: let $q = N/p$; **return** 1 iff p, q are safe primes $> 2^{\lambda+1}$

Fig. 2: λ-rough group generator based on the signed quadratic residues of an RSA group [Pie19].

In Figure 1, we present the non-interactive version of Pietrzak's argument [Pie19, BBF18], compiled from the interactive one through the Fiat-Shamir heuristic. For simplicity, we assume T is a power of two, but the protocol can be adjusted to handle arbitrary T values, as mentioned in the original publication. The verifier needs to perform $\mathcal{O}(\log_2(T))$ small exponentiations in G, whereas, as presented, the prover must compute $\mathcal{O}(T)$ group squarings. However, if the prover has also solved the underling timelock puzzle (i.e. computed h from G, T and g), and stored some relevant values during the computation, the proof can be completed in $\mathcal{O}(\sqrt{T})$ squarings [Pie19, BBF18].

Pietrzak's argument is *complete* in the following sense, which can be easily shown by induction on t. For every G, $g \in G$, $T = 2^t \in \mathbb{N}$ and $\pi \leftarrow \mathsf{Pietrzak.Prove}(G, T, g, g^{2^T})$, it holds $\mathsf{Pietrzak.Verify}(G, T, g, g^{2^T}, \pi) = 1$.

Theorem 1 (Pietrzak's Soundness). *Let $\mathsf{H} : \{0,1\}^* \to \{1, \ldots, 2^\lambda\}$ be a random oracle. For every (unbounded) algorithm $\mathcal{A}$ making at most Q queries to H, and every $\lambda, T \in \mathbb{N}$:*

$$\Pr\left[(G, g, h, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(1^\lambda, T) : \begin{array}{l} G \text{ is } \lambda\text{-rough} \wedge g^{2^T} \neq h \\ \wedge\ \mathsf{Pietrzak.Verify}(G, T, g, h, \pi) = 1 \end{array}\right] \leq \frac{Q}{2^\lambda} \ .$$

Building λ-rough groups. A possible way to build a λ-rough group is to consider a composite-order elliptic-curve whose order is λ-rough. However, the group order would not be hidden in this case. A simple alternative, suggested by Pietrzak [Pie19], is to work over the signed quadratic residues [FS97, HK09] of an RSA group whose modulus is the product of two safe primes. More concretely, let $N \coloneqq p \cdot q$ for safe primes p and q and define:

$$QR_N \coloneqq \{x \in \mathbb{Z}_N^* \ : \ \exists r \in \mathbb{Z}_N^* . \ r^2 =_N x\} \qquad QR_N^+ \coloneqq \{|x| \in \mathbb{Z}_N^* \ : \ x \in QR_N\} \ ,$$

where $|x|$ denotes the absolute value of integer x, when the elements of $\mathbb{Z}_N^*$ are represented as integers in the range $[-(N-1)/2, (N-1)/2]$. QR_N^+ is a cyclic group of order $\varphi(N)/4$ for group law $\circ$ defined as $x \circ y \coloneqq |x \cdot y \bmod N|$. Furthermore, membership of an element x in QR_N^+ can be efficiently tested by checking that $x \in [1, (N-1)/2]$ and that x has Jacobi symbol $+1$. For group trapdoor τ we can simply select one of the two prime factors of N (see Figure 2).

3 Timed commitments

In this section, we present our new definition of timed commitments and their security properties.

Definition 4. *A timed commitment scheme is a tuple of PPT algorithms:*

- $\mathsf{Setup}(1^\lambda) \to pp$*, on input the security parameter λ, outputs some public parameters pp.*

- $\mathsf{Commit}(1^\lambda, pp, T, m) \to \psi$, *on input* λ, *a time-bound* $T \in \mathbb{N}$, *a message* $m \in \{0,1\}^*$, *and some public parameters, outputs a commitment* ψ.
- $\mathsf{Open\&Prove}(T, \psi) \to (m, \pi)$, *on input a time-bound and a commitment, outputs a message (or* $\perp$*) and a proof* π.
- $\mathsf{Verify}(T, \psi, m, \pi) \to 1/0$, *on input a time-bound, a commitment, a message/*$\perp$*, and a proof, outputs a bit (*1 *representing acceptance and* 0 *representing rejection).*

The Setup algorithm is optional, a construction that does not require such algorithm is called *setup-free*. Algorithms $\mathsf{Open\&Prove}$ (on its first output) and Verify are deterministic. Furthermore, algorithms Commit and Verify run in time $\mathcal{O}(\log T)$, whereas $\mathsf{Open\&Prove}$ has time complexity $\Theta(T)$.

Valid commitments. We denote by $\mathsf{Open}(T, \psi)$ the first output of $\mathsf{Open\&Prove}$. Furthermore, we say that a commitment ψ is *valid* with respect to time-bound T if $\mathsf{Open}(T, \psi) \neq \perp$. Otherwise, we say the commitment is *invalid* (with respect to T).

A timed commitment scheme is *correct* if any honestly generated commitment opens to the message that was committed and if the verification of the proof produced by an honest execution of $\mathsf{Open\&Prove}$ is always successful.

Definition 5 (Correctness). *A timed commitment scheme is* correct *if* $\forall\, \lambda, T \in \mathbb{N}$ *and all* $pp \leftarrow \mathsf{Setup}(1^\lambda)$*:*

1. *for every* $m \in \{0,1\}^*$, $\Pr\left[\psi \leftarrow \mathsf{Commit}(1^\lambda, pp, T, m);\ m' \leftarrow \mathsf{Open}(T, \psi) : m = m'\right] = 1$,
2. *for every (adversarially generated) commitment* ψ,

$$\Pr\left[(m, \pi) \leftarrow \mathsf{Open\&Prove}(T, \psi) : \mathsf{Verify}(T, \psi, m, \pi) = 1\right] = 1.$$

A timed commitment scheme is *sound* if it is computationally unfeasible to produce false proofs of opening, even if the adversary is allowed to choose the commitment and the wrong opening message.

Definition 6 (Soundness). *A timed commitment scheme scheme is* sound *if for every PPT algorithm* $\mathcal{A}$*, and for every* $T \in \mathbb{N}$*, the following probability is negligible in* λ*:*

$$\Pr\left[(\psi, m^*, \pi^*) \leftarrow \mathcal{A}(1^\lambda, T) \ :\ \mathsf{Verify}(T, \psi, m^*, \pi^*) = 1 \ \wedge\ m^* \neq \mathsf{Open}(T, \psi)\right].$$

Finally, a timed commitment scheme is *sequential* if it is unfeasible to reveal any information about the committed message in less than the prescribed time T, even using parallelization.

Definition 7 (Sequentiality). *A timed commitment scheme is* (σ, ρ)-sequential *if for any pair of randomized algorithms* $\mathcal{A}_1$*, running in total time* $\mathcal{O}(\mathsf{poly}(\lambda, T))$*, and* $\mathcal{A}_2$*, running in parallel time* $\sigma(T)$ *on at most* $\rho(T)$ *parallel processors, and for any* λ*, the following difference is negligible in* λ*:*

$$\Pr\left[\begin{array}{r} (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, T) \\ b \leftarrow \{0,1\};\ pp \leftarrow \mathsf{Setup}(1^\lambda);\ \psi \leftarrow \mathsf{Commit}(1^\lambda, pp, T, m_b) \ :\ b = b' \\ b' \leftarrow \mathcal{A}_2(\mathsf{st}, \psi) \end{array}\right] - \frac{1}{2}\ .$$

where it is required that $\mathcal{A}_1$ *produce two messages of the same length.*

4 Constructions

In this section, we present two very simple an efficient timed commitment schemes, based on hidden order groups, with different trade-offs. Our first construction does not require a setup phase, but it requires using a rough-order subgroup of an RSA group. Our second construction can be instantiated over any group of hidden order, but it requires a setup phase for every desired value of T.

Both constructions are extensions of the timelock puzzles by Rivest et al. [RSW96] equipped with mechanisms for public verifiability (from VDF technology [BBBF18, Wes19, Pie19]). Consequently, sequentiality in both constructions follows from the assumption that iterated squaring over the relevant group is a sequential problem and the security of the encryption scheme (which only needs to be 1-time CPA secure, see Definition 2) and the security of the key derivation function.

4.1 First construction: a setup-free timed commitment

The main idea behind this scheme is that the party who commits chooses the group and will know the group order. This allows them perform iterated squaring efficiently over the group. However, the group must be λ-rough (see Figure 2) to guarantee that Pietrzak's argument is sound, even against parties who know the group order. To ensure that the party who commits does not cheat and indeed chooses a group which is λ-rough, they are required to include a witness of the λ-roughness of the group in the committed message. A commitment will only be considered valid if its opened message indeed contains a witness that asserts the λ-roughness of the group.

Theorem 2. *The timed commitment scheme from Figure 3 is correct and sound in the ROM.*

Proof. Correctness can be checked by inspection. Soundness is a direct consequence of Lemmas 1 and 2 which, together, imply a stronger notion of soundness (against unbounded adversaries). □

Lemma 1. *For every $\lambda, T \in \mathbb{N}$, every $m^* \neq \bot$, ψ s.t. $\mathsf{Open}(T, \psi) \neq m^*$, and all π^*, $\mathsf{Verify}(T, \psi, m^*, \pi^*) = 0$.*

Proof. Let ψ be (G, g, ct) and parse π^* as (h^*, π_P^*). Since $m^* \neq \bot$, if $\mathsf{Verify}(T, \psi, m^*, \pi^*)$ returns 1, then the conditional in step 3 must be satisfied, thus $\mathsf{Dec}_{\mathsf{KDF}(h^*)}(\mathsf{ct}) = (\tau^* \| m^*)$ with $\lambda\text{-}\mathsf{rough}(G, \tau^*) = 1$ and $g \in G$. Also, for Verify to return 1 in step 5, given that the group order is correct, it must hold $g^{2^T} = h^*$. Under these conditions, it is clear that $\mathsf{Open}(T, \psi)$ will return m^*, violating the Lemma precondition. □

Lemma 2. *For every probabilistic (unbounded) algorithm $\mathcal{A}$ making at most Q queries to its random oracle [4], all $\lambda, T \in \mathbb{N}$ and every ψ s.t. $\mathsf{Open}(T, \psi) \neq \bot$, $\Pr[\pi^* \leftarrow \mathcal{A}(T, \psi) : \mathsf{Verify}(T, \psi, \bot, \pi^*) = 1] \leq Q/2^\lambda$.*

Proof. Let ψ be (G, g, ct). Since $\mathsf{Open}(T, \psi) \neq \bot$, the $\mathsf{Open\&Prove}$ algorithm from Figure 3 returned a value in step 3, which implies that $g \in G$ and

$$\mathsf{Dec}_{\mathsf{KDF}(g^{2^T})}(\mathsf{ct}) = (\tau \,\|\, \_) \text{ with } \lambda\text{-}\mathsf{rough}(G, \tau) = 1 \ . \tag{1}$$

For Verify to return 1 when $m = \bot$, $\mathcal{A}$ must produce $\pi^* := (h^*, \pi_P^*)$ such that:

$$\mathsf{Pietrzak.Verify}(G, T, g, h^*, \pi_P^*) = 1 \ \wedge \ \mathsf{Dec}_{\mathsf{KDF}(h^*)}(\mathsf{ct}) = (\tau^* \,\|\, \_) \text{ with } \lambda\text{-}\mathsf{rough}(G, \tau^*) = 0 \ .$$

Note that equation (1) implies that for the second clause in the above conjunction to hold, it must be $\mathsf{KDF}(h^*) \neq \mathsf{KDF}(g^{2^T})$. Assuming KDF is an injective function we have $h^* \neq g^{2^T}$. However, in virtue of Theorem 1, if $h^* \neq g^{2^T}$ the probability that an adversary performing at most Q queries to the RO produces π_P^* such that $\mathsf{Pietrzak.Verify}(G, T, g, h^*, \pi_P^*) = 1$, for a λ-rough group G (as ensured by (1)), is upper-bounded by $Q/2^\lambda$. □

$\mathsf{Commit}(1^\lambda, T, m)$:

1: sample $(G, \tau) \leftarrow \mathcal{G}(1^\lambda)$ and $g \leftarrow G$ ▷ See Figure 2

2: set $e := (2^T \bmod \mathsf{order}(G, \tau))$, compute[a] $k := \mathsf{KDF}(g^e)$ and $\mathsf{ct} \leftarrow \mathsf{Enc}_k(\tau \,\|\, m)$

3: **return** $\psi := (G, g, \mathsf{ct})$

$\mathsf{Open\&Prove}(T, \psi := (G, g, \mathsf{ct}))$:

1: compute $h := g^{2^T}$, $k := \mathsf{KDF}(h)$, and $\mathsf{pt} := \mathsf{Dec}_k(\mathsf{ct})$

2: compute $\pi_P \leftarrow \mathsf{Pietrzak.Prove}(G, T, g, h)$ and set $\pi := (h, \pi_P)$

3: **if** $\bot \neq \mathsf{pt} = (\tau \,\|\, m)$ and $\lambda\text{-}\mathsf{rough}(G, \tau) = 1$ and $g \in G$ **then return** (m, π)

4: **else return** $(\bot, \pi)$

$\mathsf{Verify}(T, \psi := (G, g, \mathsf{ct}), m, \pi := (h, \pi_P))$:

1: compute $k := \mathsf{KDF}(h)$ and $\mathsf{pt} := \mathsf{Dec}_k(\mathsf{ct})$

2: **if** $m \neq \bot$ **then**

3: **if** $\mathsf{pt} = (\tau \,\|\, m')$ with $m' = m$ and $\lambda\text{-}\mathsf{rough}(G, \tau) = 1$ and $g \in G$ **then**

4: let $e := (2^T \bmod \mathsf{order}(G, \tau))$ and compute $h' := g^e$

5: **return** $(h' = h)$

6: **else return** 0

7: **else return** $\mathsf{Pietrzak.Verify}(G, T, g, h, \pi_P) = 1$ and $(\mathsf{pt} \neq (\tau \,\|\, \_)$ or $\lambda\text{-}\mathsf{rough}(G, \tau) = 0)$ ▷ $m = \bot$

Fig. 3: Setup-free timed commitment scheme.

[a] KDF is a key derivation function from G elements to the key space of Enc.

4.2 Second construction

The main idea behind our second construction (described Figure 4) is to use a group whose order is hidden to everyone, even the party who commits. This construction is less limited than our previous one in the type of group, as normal RSA groups or class groups can be used. Furthermore, this construction has the nice property that the committer can create a proof of opening very efficiently.

In order for the commitment algorithm to be efficient, we will precompute a timelock puzzle-solution pair (x, y) for the desire time bound T. We will also precompute a Pietrzak argument of the validity of the pair. (In fact, we could use any VDF argument which is compatible with the iterated squaring function, e.g. Wesolowski's argument [Wes19], as in this case the group order is not known to anyone). Such pair can be rerandomized as (x^r, y^r) for any $r \in \mathbb{N}$, leading to a new valid pair. (This technique has been previously used in the literature). However, updating the VDF argument for the new rerandomized pair does not seem possible. To resolve this problem, we will accept the validity of a puzzle-solution pair if it is "linked" to an equivalent pair for which a valid VDF argument exists.

Theorem 3. *The timed commitment scheme from Figure 4 is correct and sound.*

[4] The random oracle is involved in Pietrzak's non-interactive argument.

$\underline{\mathsf{Setup}(1^\lambda)}$:

1: sample a group G of hidden order, using security parameter λ

2: **for** a range of T_i **do**

3: sample $g_i \leftarrow G$ and compute $h_i := g^{2^{T_i}}$

4: generate $\pi_i^{\mathsf{VDF}} \leftarrow \mathsf{VDF.Prove}(G, T_i, g_i, h_i)$

5: **return** $(G, \{g_i, h_i, \pi_i^{\mathsf{VDF}}\}_i)$

$\underline{\mathsf{Commit}(1^\lambda, pp := (G, \{g_i, h_i, \pi_i^{\mathsf{VDF}}\}_i), T, m)}$:

1: Let $(g, h, \pi^{\mathsf{VDF}})$ be the tuple in pp relative to T

2: sample $r \in \mathbb{N}$ with enough entropy, e.g. uniformly in $[1, 2^\lambda]$, and compute $(g^*, h^*) := (g^r, h^r)$

3: compute $\mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{KDF}(h^*)}(m)$ and **return** $\psi := (G, g^*, \mathsf{ct})$.

$\underline{\mathsf{Open\&Prove}(T, \psi := (G, g^*, \mathsf{ct}))}$:

1: compute $h^* := {g^*}^{2^T}$, and $\pi^*_{\mathsf{VDF}} \leftarrow \mathsf{VDF.Prove}(G, T, g^*, h^*)$

2: set $m := \mathsf{Dec}_{\mathsf{KDF}(h^*)}(\mathsf{ct})$ and $\pi := (g^*, h^*, \pi^*_{\mathsf{VDF}}, 1)$[a]

3: **return** (m, π)

$\underline{\mathsf{Verify}(T, \psi := (G, g^*, \mathsf{ct}), m, \pi := (g, h, \pi_{\mathsf{VDF}}, s))}$:

1: **return** $g^* = g^s \ \wedge \ \mathsf{VDF.Verify}(G, T, g, h, \pi_{\mathsf{VDF}}) = 1 \ \wedge \ \mathsf{Dec}_{\mathsf{KDF}(h^s)}(\mathsf{ct}) = m$

Fig. 4: Second construction of a timed commitment scheme.

[a] The chest creator can create a valid proof very efficiently (skipping the above VDF proving, by reusing the existing precomputed proof) if they keep the randomness r used during the committing phase. Namely, they can set $\pi = (g, h, \pi_{\mathsf{VDF}}, r)$, where $(g, h, \pi_{\mathsf{VDF}})$ is the tuple from pp relative to T.

Proof. Correctness can be checked by inspection. Consider a PPT adversary $\mathcal{A}$ against the soundness of the scheme. We build a PPT adversary $\mathcal{B}$ against the soundness of the underlying VDF argument. On input G and $T \in \mathbb{N}$, $\mathcal{B}$ forwards them to $\mathcal{A}$, who will produce a triple (ψ, m, π). $\mathcal{B}$ will then parse π as $(g, h, \pi_{\mathsf{VDF}}, s)$ and return $(g, h, \pi_{\mathsf{VDF}})$. We argue that if $\mathcal{A}$ is successful, then $\mathcal{B}$ is successful (in their respective games).

Assume $\mathcal{A}$ is successful, i.e. it holds both that $\mathsf{Verify}(T, \psi, m, \pi) = 1$ and $m \neq \mathsf{Open}(T, \psi)$, or equivalently, with $\psi := (G, g^*, \mathsf{ct})$, we have that $g^* = g^s \ \wedge \ \mathsf{VDF.Verify}(G, T, g, h, \pi_{\mathsf{VDF}}) = 1 \ \wedge \ \mathsf{Dec}_{\mathsf{KDF}(h^s)}(\mathsf{ct}) = m$ and $m \neq \mathsf{Dec}_{\mathsf{KDF}({g^*}^{2^T})}(\mathsf{ct})$.

Now, assume for a moment that g^{2^T} equals h. Given $g^* = g^s$, we would have ${g^*}^{2^T} = {g^s}^{2^T} = (g^{2^T})^s = h^s$. Therefore, since Dec is deterministic, $\mathsf{Dec}_{\mathsf{KDF}({g^*}^{2^T})}(\mathsf{ct}) = \mathsf{Dec}_{\mathsf{KDF}(h^s)}(\mathsf{ct})$. But this is a contradiction since the left-hand side of the previous equality is different from m, whereas the right-hand side equals m. Consequently, we conclude that g^{2^T} and h are different, which combined with the fact that $\mathsf{VDF.Verify}(G, T, g, h, \pi_{\mathsf{VDF}}) = 1$ imply that $\mathcal{B}$ is successful too, as desired. □

References

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

BBF18. Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint.iacr.org/2018/712.

BN00. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Heidelberg, August 2000.

CJ22. Peter Chvojka and Tibor Jager. Simple, fast, efficient, and tightly-secure non-malleable non-interactive timed commitments. Cryptology ePrint Archive, Report 2022/1498, 2022. https://eprint.iacr.org/2022/1498.

FS97. Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for RSA and rabin bits. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 267–279. Springer, Heidelberg, May 1997.

HK09. Dennis Hofheinz and Eike Kiltz. The group of signed quadratic residues and applications. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 637–653. Springer, Heidelberg, August 2009.

KLX20. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, November 2020.

MA22. Yacov Manevich and Adi Akavia. Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments. Cryptology ePrint Archive, Report 2022/717, 2022. https://eprint.iacr.org/2022/717.

Pie19. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.

RSW96. Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.

TCLM21. Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. Efficient CCA timed commitments in class groups. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2663–2684. ACM Press, November 2021.

Wes19. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.