Homomorphic Time-Lock Puzzles and Applications

Giulio Malavolta[1⋆] and Sri Aravinda Krishnan Thyagarajan[2]

[1] Carnegie Mellon University
[2] Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. Time-lock puzzles allow one to encrypt messages for the future, by efficiently generating a puzzle with a solution s that remains hidden until time $\mathcal{T}$ has elapsed. The solution is required to be concealed from the eyes of any algorithm running in (parallel) time less than $\mathcal{T}$.
We put forth the concept of *homomorphic time-lock puzzles*, where one can evaluate functions over puzzles without solving them, i.e., one can manipulate a set of puzzles with solutions $(s_1, \ldots, s_n)$ to obtain a puzzle that solves to $f(s_1, \ldots, s_n)$, for any function f. We propose candidate constructions under concrete cryptographic assumptions for different classes of functions. Then we show how homomorphic time-lock puzzles overcome the limitations of classical time-lock puzzles by proposing new protocols for applications of interest, such as e-voting, multi-party coin flipping, and fair contract signing.

1 Introduction

Time-lock puzzles [30] allow one to encapsulate messages for a precise amount of time or, equivalently, to encrypt messages for the future. On input a secret s and a hardness parameter $\mathcal{T}$, the puzzle generation algorithm allows one to compute a Z such that s can be recovered only after time $\mathcal{T}$. Time-lock puzzles are characterized by the following properties.

- *Fast puzzle generation:* The time needed to generate a puzzle is much shorter than $\mathcal{T}$. This is crucial when secrets are hidden for a long time, e.g., 10 years.
- *Security against parallel algorithms:* The secret s is hidden for circuits of depth less than $\mathcal{T}$, regardless of their size.

The latter can be seen as a more fine-grained notion of the classical semantic security [19], where simply lowering the security parameter may enable faster algorithms that exploit massive parallelization to solve the puzzle. Note that ignoring either of the above properties makes the problem trivial since it can be either solved with standard probabilistic encryption or any inherently sequential computation (such as repeated hashing). Applications of time-lock puzzles include sealed-bid auctions [30], fair contract signing [6], zero-knowledge arguments [12], and non-malleable commitments [23], to mention a few.

To compensate for the absence of a trusted party, time-lock puzzles force the decrypter to perform a long computation before being able to recover the secret. When time-lock puzzles are deployed within large scale protocols, this slight drawback is magnified and parties may incur in a significant computational burden. While performing *some* computation is clearly unavoidable, this effort should not become the bottleneck of the protocol. To the best of our knowledge, there is currently no solution to mitigate this problem.

1.1 Limitations of Time-Lock Puzzles

To illustrate the aforementioned limitations of time-lock puzzles, we consider the scenario of e-voting in the absence of a trusted authority, one of the motivating examples for the usage of the primitive. Throughout the following discussion we assume that the voters have access to a public and append-only bulletin board, e.g., a blockchain, and we will not consider the privacy of the votes nor their authenticity. Both problems are

⋆ Part of the work done while at Friedrich-Alexander-Universität Erlangen-Nürnberg.

well studied and can be dealt with using standard techniques, e.g., unlinkable transactions and anonymous credentials. Instead, we are going to focus on constructing a system that allows a large set of voters to cast their preference without any bias.

If one were to assume a trusted administrator, then the voters could simply encrypt their preference and let the administrator count and announce the result. However, the absence of trusted authorities makes the problem non-trivial. The standard approach to avoid voters being biased by the current majority is to divide the protocol in two phases: In the *voting phase* the voters commit to their vote and post the commitment on the bulletin board. In the *counting phase*, new commitments are ignored, and voters are asked to reveal their openings, which makes it possible to compute and announce the result of the election.

This however leaves open the question of how to handle users who send valid commitments in the first phase but fail to reveal their openings in the second. One could either (i) repeat the voting phase or (ii) ignore such "unopened" votes. Repeating the voting process could empower an attacker to successfully mount a denial-of-service attack at essentially no cost. On the other hand, the latter solution might be exploited to manipulate the final outcome: An attacker controlling the network traffic might block those openings corresponding to an unwanted candidate, thereby generating a bias towards a the preferred side.

Time-lock puzzles elegantly resolve this by replacing commitments as the hiding mechanism for the votes. The votes of those users who fail to publish their coins (i.e., reveal their vote) can be simply determined by solving their time-lock puzzles. Setting the hardness parameter $\mathcal{T}$ to be a safe amount longer than the voting phase makes sure that the votes are kept secret until such a phase is over, thereby avoiding any bias. Unfortunately, this solution does not come without additional costs: Consider what happens when a large amount of voters, say 100.000, fail to open their puzzles. Then the computation of the election winner tally requires brute-forcing those puzzles, which means that a massive amount of (parallel) computation is needed in order to complete the election within reasonable time. Taking into account the typical number of voters for an election which is usually in the range of millions, it is safe to say that the problem is of practical relevance.

We stress that, even though e-voting exemplifies well the scalability issues of time-lock puzzles, it is certainly not the only scenario where they emerge. Essentially any other application that involves a large number of users (e.g., sealed bid auctions or multi-party coin flipping), encounters similar problems. We conjecture that such constraints constitute one of the main obstacles that so far prevented the large scale adoption of time-lock puzzles.

1.2 Our Solution

Put in different words, the main shortcoming of time-lock puzzle-based solutions is that one needs to solve (brute-force) many puzzles before computing some function over the embedded secrets. What if we could (homomorphically) evaluate the function first and then solve a *single puzzle* containing the function output? This would dramatically reduce the computational burden of time-lock puzzle-based protocols. Consider the e-voting example as described above: To compute the election winner one could homomorphically evaluate the corresponding circuit over the puzzles and then solve *a single puzzle*, regardless of the number of offline voters.

Motivated by this question, we propose the notion of *Homomorphic Time-Lock Puzzles* (HTLP): Loosely speaking, an HTLP is an augmented time-lock puzzle that allows anyone to evaluate a circuit C over sets of puzzles $(Z_1, \ldots, Z_n)$ homomorphically, without necessarily knowing the secret messages $(s_1, \ldots, s_n)$ encapsulated within these puzzles. The resulting output (which is also a puzzle) contains the circuit output $C(s_1, \ldots, s_n)$ and the timing hardness of this puzzle does not depend on the size of the circuit C that was evaluated (compactness). We stress that the compactness of the evaluation algorithm is a crucial requirement for HTLP (as it is the case for fully-homomorphic encryption [16]): If we were to ignore it, then the trivial solution of solving the puzzles $(Z_1, \ldots, Z_n)$ and then evaluating $\mathcal{C}$ over the secrets would suffice.

In this work we put forward the concept of HTLPs and we formally characterize their security guarantees. We then propose several schemes that support the homomorphic evaluation of different classes of circuits and we demonstrate their usefulness by presenting several concrete applications.

1.3 Technical Overview

Towards instantiating HTLPs, our starting point is the classical construction of Rivest, Shamir, and Wagner [30], whose hardness is rooted in the (conjectured) inherent sequentiality of squaring in finite fields. Let $N = p \cdot q$ be an RSA integer, a time-lock puzzle for a secret s and for a time $\mathcal{T}$ consists of the tuple

$$(N, \mathcal{T}, x, x^{2^{\mathcal{T}}} \cdot k, \mathsf{Enc}(k, s))$$

where (x, k) are uniformly sampled elements from $\mathbb{Z}_N^*$, and $\mathsf{Enc}(k, s)$ is a symmetric encryption of the secret s. Note that knowing the group order $\varphi(N)$ allows one to efficiently compute the term $x^{2^{\mathcal{T}}}$ by reducing $2^{\mathcal{T}}$ modulo $\varphi(N)$ first. On the other hand the decrypter has to perform $\mathcal{T}$-many squarings before recovering the key k. Here the hybrid encryption approach breaks the structure of the group, and therefore the scheme has no homomorphic properties.

Linearly Homomorphic. Our first observation is that the term $x^{2^{\mathcal{T}}}$ acts essentially as a one-time pad and we can choose a more structured embedding that admits an efficiently computable homomorphism. We follow the blueprint of Paillier [28], i.e., we exploit the fact that the group $\mathbb{Z}_{N^2}^*$ can be written as the product of the group generated by $(1+N)$, which has order N, and the group of N-th residues $\{x^N : x \in \mathbb{Z}_N^*\}$, which has order $\varphi(N)$. Consider the following (flawed) attempt to construct HTLPs for linear functions:

$$(N, \mathcal{T}, x, x^{N \cdot 2^{\mathcal{T}}} \cdot (1+N)^s),$$

for a random $x \in \mathbb{Z}_N^*$. Assume for the moment that N is fixed across all puzzles, then the scheme is clearly linearly homomorphic as shown below:

$$(N, \mathcal{T}, x \cdot y, x^{N \cdot 2^{\mathcal{T}}} \cdot y^{N \cdot 2^{\mathcal{T}}} \cdot (1+N)^s \cdot (1+N)^{s'}) = (N, \mathcal{T}, (x \cdot y), (x \cdot y)^{N \cdot 2^{\mathcal{T}}} \cdot (1+N)^{s+s'}).$$

Observe that the time needed to homomorphically add secrets is independent of $\mathcal{T}$. Further recall that the group generated by $(1 + N)$ admits a polynomial-time algorithm to compute discrete logarithms, so recovering the output is easy once $x^{N \cdot 2^{\mathcal{T}}}$ is computed. Unfortunately there are two major issues with the current scheme: (i) If N is shared across all users who also generated them, then everybody potentially knows the factorization of N (and therefore $\varphi(N)$), which is a problem for security, and (ii) the blinding factor $x^{N \cdot 2^{\mathcal{T}}}$ is trivially distinguishable from a uniform element in $\mathbb{Z}_N^*$ as the Jacobi symbol of $x^{N \cdot 2^{\mathcal{T}}}$ is always $+1$. The latter issue can be easily countered by restricting the random choice to those elements in $\mathbb{Z}_N^*$ whose Jacobi symbol is equal to $+1$. Our idea to sidestep the former limitation is to use the random self-reducibility of the problem: In our augmented scheme, the tuple $(N, x, x^{2^{\mathcal{T}}})$, where x is a random element of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$, is fixed in a setup phase. A freshly-looking HTLP can now be computed as

$$(N, \mathcal{T}, x^r, (x^{N \cdot 2^{\mathcal{T}}})^r \cdot (1+N)^s) = (N, \mathcal{T}, y, y^{N \cdot 2^{\mathcal{T}}} \cdot (1+N)^s),$$

where r is uniformly sampled from $\{1, \ldots, N^2\}$, whose distribution (modulo $\varphi(N)$) is statistically close to sampling from $\{1, \ldots, \varphi(N)\}$. Note that the newly generated puzzle is correctly distributed and the knowledge of $\varphi(N)$ is not needed to compute it. It can be shown that the scheme is an HTLP for linear functions, assuming the inherent sequentiality of squaring modulo N and other standard intractability assumptions over hidden-order groups.

Multiplicatively Homomorphic. Armed with the tools discussed above, we can easily switch the message encoding to obtain HTLPs that supports the evaluation of multiplication gates. This is done by adapting the scheme of above to a Diffie-Hellman structure in a natural way: Given that the tuple $(N, x, x^{2^{\mathcal{T}}})$ is fixed in a setup phase, a puzzle to encapsulate a secret $s \in \mathbb{J}_N$ (where $\mathbb{J}_N$ is the subgroup of $\mathbb{Z}_N^*$ whose elements have Jacobi symbol $+1$) is generated as

$$(N, \mathcal{T}, x^r, (x^{2^{\mathcal{T}}})^r \cdot s)$$

for some uniformly chosen r. The procedure to recover the puzzle is essentially unchanged, except that now all the operations are performed in the subgroup $\mathbb{J}_N$. Clearly, there is no need to compute any discrete logarithm since s is already in its plain form. In [10] it was shown that the decisional Diffie-Hellman (DDH) assumption over $\mathbb{J}_N$ is implied by the DDH assumption over $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ and the quadratic residuosity assumption over $\mathbb{Z}_N^*$. Thus the security of our scheme follows from the same set of hard problems (in addition to assuming the sequential nature of squaring modulo N).

Fully Homomorphic. The schemes constructed above support the homomorphic evaluation of some restricted classes of functions over the secrets. The next natural question is whether there exists an HTLP for *any* polynomially-computable function. It seems like the techniques developed so far are not very helpful in this context since constructing homomorphic encryption from RSA groups (and related assumptions) has been an elusive task so far. For this reason we turn our attention to constructions based on indistinguishability obfuscation [14]. The scheme that we obtain shall be interpreted as a feasibility result. We leave constructing HTLPs for any function without the aid of obfuscation as a fascinating open problem. Our candidate solution follows the blueprint of the fully-homomorphic encryption (FHE) scheme from [9]. Omitting most of the technicalities, their FHE is constructed from standard public-key encryption by obfuscating a program that decrypts two input ciphertexts, computes a NAND gate over the messages, and re-encrypts the output. This approach allows one to construct FHE without relying on circular assumptions, since the obfuscated program can evaluate circuits of any depth without growing in size.

At a first glance, this strategy does not seem to translate directly to the time-lock puzzle settings, since puzzles do not necessarily have a trapdoor that allows one to efficiently recover the secret (see, e.g., the construction from [2]). Instead of replacing the public key encryption, our scheme *augments it* by additionally time-locking the message: The puzzle consists of a tuple (c, Z), where the ciphertext c and any (non-homomorphic) time-lock puzzle Z encode the same message. To open it, one simply ignores c and solves Z. To support homomorphic computations, we obfuscate a program that takes as input two puzzles (c_0, Z_0) and (c_1, Z_1), decrypts c_0 and c_1, computes the NAND of the messages and produces a fresh pair (c', Z') encoding the output bit. Note that, although the program discards Z_0 and Z_1, the output puzzle is still well-formed. Such a program is obfuscated in the setup phase and it is made available to all parties.

Extensions. The constructions presented above constitute the backbone of our results, but there are still a few shortcomings that need to be addressed in order to enjoy all advantages of HTLPs. For example, all of the schemes (as described above) require a trusted setup that needs to be re-initialized once time $\mathcal{T}$ has passed. We show that this is in fact not necessary for our RSA-based schemes and that the common reference string $(N, x, x^{2^{\mathcal{T}}})$ can be fixed once and for all in a *one-time setup*, assuming a mildly stronger version of the sequential squaring problem. We also show how to compute homomorphic operations over puzzles generated under different hardness parameters and we explore the feasibility of a non-trusted (public-coin) setup. Finally, we present a semi-compact HTLP for branching programs (a superclass of NC1), where the size of the evaluated puzzle grows with the length of the program but not with its size.

1.4 Applications

We substantiate our claims with concrete examples of scenarios where HTLPs are useful. Due to the different nature of our constructions, we focus on how to exploit our efficient (RSA-based) schemes to build applications of interest.

E-Voting and Sealed Bid Auctions over Blockchains. We consider the settings where n voters choose one among m candidates and we assume that $n \gg m$. In our protocol, each voter generates a vector of m *linearly-homomorphic* puzzles $(Z_1, \ldots, Z_m)$ encapsulating 0, except for the j-th puzzle Z_j that encodes 1, where j is the index of the preferred candidate.[3] The vector of each voter is made available to all parties (by, e.g., posting it on a blockchain) during the voting phase. Afterwards, the outcome of the election can be determined by simply summing up all vectors and opening the resulting m puzzles. The resulting vector

[3] We implicitly assume that all puzzles are honestly generated, which can be enforced with standard cryptographic tools.

will contain the amount of votes per candidate and the winner can then be easily determined. Note that this is a public operation and therefore there is no need for a trusted tallying authority. Furthermore, the computational effort needed to determine the result of the election is that of solving m puzzles, regardless on the amount of voters. The typical values for m are in the order of tens, which corresponds to a manageable amount of computation for essentially any machine. This is a significant improvement with respect to the original solution that required the opening of potentially hundreds of thousands puzzles.

Similar techniques can be used to design a sealed bid auction protocol: Each bidder time-locks its bid and the index corresponding to the highest bidder is homomorphically computed over the puzzles. The winner of the auction can be determined by solving a single puzzle. Unfortunately the resulting protocol is not yet practical since the circuit being evaluated exceeds the capability of linear functions and requires *fully-homomorphic* time-lock puzzles.

Multi-Party Coin Flipping. Coin flipping protocols are one of the classical problems in cryptography [3] and have recently found applications in real-life cryptocurrencies [22]. The security required by an n-party coin flipping protocol is that $n-1$ colluding parties should not be able to bias the final outcome. Boneh and Naor [6] proposed a solution for coin flipping among two parties based on time-lock puzzles. However, naively extending their protocol to the multi-party setting suffers from predictable drawbacks: The computational effort of the participants is proportional to the amount of parties that do not reveal their random coins. This becomes very significant when coin flipping protocols are executed on a large scale (e.g., thousands of participants). Using *linearly-homomorphic* time-lock puzzles we obtain a very simple solution to this problem. Each participant encapsulates a random bit for a safe amount of time and broadcasts it to all parties. Then each party homomorphically add all puzzles, without the need for further interactions. Solving the resulting output puzzle and isolating the least significant bit of the solution gives us an unbiased coin.

Multi-Party Contract Signing. Consider the scenario where n mutually distrusting parties want to exchange signatures on a document. Boneh and Naor [6] proposed a protocol for fair exchange of signatures based on time-lock puzzles. The protocol proceeds in rounds where each party generates a time-lock puzzle of their signature and broadcasts it. When all signatures are published, the protocol repeats except that the hardness parameter of the time-lock puzzle is halved. The protocol is strongly fair in the sense that the work required to recover the signatures by all parties differs at most by a factor of (roughly) 2.

Observe that if at any round *any* party fails to broadcast its puzzle, then all other parties need to solve *all* the puzzles ($(n-1)$-many) from the previous round to learn the signatures necessary for the validity of the contract. Our *multiplicatively-homomorphic* time-lock puzzles can be plugged in this protocol to solve exactly this issue. More specifically, we can leverage a recent result on RSA-aggregate signatures [20], where Hohenberger and Waters proposed a scheme where signatures live in $\mathbb{QR}_N$, where N is fixed in the setup, and can be aggregated by simply multiplying them modulo N. Recall that $\mathbb{QR}_N$ is a subgroup fo $\mathbb{J}_N$ and therefore signatures encapsulated in our HTLP can be aggregated homomorphically.

Equipped with this tool, we can simply replace the time-lock puzzle of Boneh and Naor with our multiplicatively homomorphic construction and combine it with the signature scheme of Hohenberger and Waters. Then, in the case that any party goes offline ahead of time, each other party can homomorphically aggregate the signatures from the previous round and then solve a *single* time-lock puzzle, regardless of the number of participants.

1.5 Related Work

Time-lock puzzles were envisioned in the seminal work by Rivest, Shamir, and Wagner [30]. Their scheme builds on the (conjectured) inherent sequentiality of repeated squaring in RSA groups. Recently, Bitanski et al. [2] proposed a different approached to construct time-lock puzzles, assuming the existence of succinct randomized encodings [1] and non-parallelizable languages. We also mention a new construction paradigm from Liu et al. [24] that combines witness encryption [15] with a reference clock, such as a blockchain.

A related but different notion is that of verifiable delay functions [4], which allow a prover to convince a verifier that a certain amount of sequential computation has been performed. The two notions are incomparable since verifiable delay functions (in general) do not allow one to encapsulate secrets and time-lock puzzles

are (in general) not efficiently verifiable. Proofs of sequential work [26] can be seen as a non-unique verifiable delay functions. Interestingly, Mahmoody et al. [25] showed a blackbox separation between time-lock puzzles and proofs of sequential work.

2 Preliminaries

We denote by $\lambda \in \mathbb{N}$ the security parameter. We we say that a function μ is negligible if it vanishes faster than any polynomial. Given two ensembles D_0 and D_1, we write $D_0 \approx_\mu D_1$ if all probabilistic polynomial-time distinguishers succeed with probability μ-close to $1/2$. Given a set U, we denote by $u \leftarrow\!\!\$\, U$ the uniform sampling from U. Recall the definition of statistical distance.

Definition 1 (Statistical Distance). *Let X and Y be two random variables over a finite set U. The statistical distance between X and Y is defined as*

$$\mathbb{SD}\left[X, Y\right] = \sum_{u \in U} \left|\Pr[X = u] - \Pr[Y = u]\right|.$$

We say that an ensemble D is ε-uniform in U if the statistical distance between D and uniformly sampling from U is at most ε. We recall the following useful lemma from [7].

Lemma 1. *Let $(n, \tilde{n}) \in \mathbb{N}^2$ and let $x \leftarrow\!\!\$\, \{1, \ldots, \tilde{n}\}$, then $x \pmod n$ is $(n/\tilde{n})$-uniform in $\mathbb{Z}_n$.*

Proof. Let $d = \tilde{n} \pmod n$, then conditioned on the event that $x \in \{1, \ldots, \tilde{n} - d\}$, it holds that $x \pmod n$ is uniformly distributed in $\mathbb{Z}_n$. Therefore $x \pmod n$ is $(d/\tilde{n}) \leq (n/\tilde{n})$-uniform.

2.1 Number Theory and Assumptions

Let $N = p \cdot q$, where p and q are random primes of equal length, we define $\mathbb{Z}_N^* \coloneqq \{x \in \mathbb{Z}_N : \gcd(x, N) = 1\}$ and $\mathbb{J}_N$ as the group of elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$ and we denote by g a generator of $\mathbb{J}_N$. Euler totient function is denoted by $\varphi(\cdot)$. We say that N is a strong RSA integer if $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also primes. Note that if N is a strong RSA integer then $\mathbb{J}_N$ is cyclic and has order $\varphi(N)/2$. Also note that a generator g for $\mathbb{J}_N$ can be found by sampling $\tilde{g} \leftarrow\!\!\$\, \mathbb{Z}_N^*$ and setting $g \coloneqq -\tilde{g}^2$ since the order of $\tilde{g}$ is either $\varphi(N)/2$ or $\varphi(N)/4$ with all but negligible probability.

We state and prove the following simple lemma, which is going to be useful in the analysis of our schemes.

Lemma 2. *For every $x \in \mathbb{N}$ and every $N \in \mathbb{N}$ it holds that*

$$x^N \ (mod\ N^2) = (x \ (mod\ N))^N \ \ (mod\ N^2).$$

Proof. Let us rewrite $x = \tilde{x} + kN$, for some k and some $\tilde{x} < N$. Then we have

$$\begin{aligned} x^N \pmod{N^2} &= (\tilde{x} + kN)^N \pmod{N^2} \\ &= \tilde{x}^N + \left(\tilde{x}^{N-1} kN\right) N + \ldots \pmod{N^2} \\ &= \tilde{x}^N \pmod{N^2} \\ &= (x \pmod N)^N \pmod{N^2}. \end{aligned}$$

Sequential Squaring. In the following we recall the intractability assumption (implicitly) introduced by Rivest, Shamir, and Wagner [30].

Assumption 1 (Sequential Squaring) *Let N be a uniform strong RSA integer, g be a generator of $\mathbb{J}_N$, and $\mathcal{T}(\cdot)$ be a polynomial. Then there exists some $0 < \varepsilon < 1$ such that for every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ who's depth is bounded from above by $\mathcal{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[b \leftarrow \mathcal{A}(N, g, \mathcal{T}(\lambda), x, y) : \begin{array}{c} x \leftarrow_\$ \mathbb{J}_N; b \leftarrow_\$ \{0,1\} \\ \text{if } b = 0 \text{ then } y \leftarrow_\$ \mathbb{J}_N \\ \text{if } b = 1 \text{ then } y \coloneqq x^{2^{\mathcal{T}(\lambda)}} \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Note that we restrict the domain of x and y to $\mathbb{J}_N$ to avoid trivial attacks where the distinguisher computes the Jacobi symbol of the group element.

Decisional Composite Residuosity. Here we recall the decisional composite residuosity (DCR) assumption as of [28].

Assumption 2 (Decisional Composite Residuosity) *Let N be a uniform strong RSA integer. Then for every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[b \leftarrow \mathcal{A}(N, y) : \begin{array}{c} x \leftarrow_\$ \mathbb{Z}_N^*; b \leftarrow_\$ \{0,1\} \\ \text{if } b = 0 \text{ then } y \leftarrow_\$ \mathbb{Z}_{N^2}^* \\ \text{if } b = 1 \text{ then } y \coloneqq x^N \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Decisional Diffie-Hellman. Here we recall the decisional composite Diffie-Hellman (DDH) assumption over $\mathbb{J}_N$ as stated in [10]. In the same work, it was shown that such a conjecture is implied by the DDH assumption over $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ and by the quadratic residuosity assumption over $\mathbb{Z}_N^*$.

Assumption 3 (Decisional Diffie-Hellman) *Let N be a uniform strong RSA integer and g be a generator of $\mathbb{J}_N$. Then for every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[b \leftarrow \mathcal{A}(N, g, g^x, g^y, g^z) : \begin{array}{c} (x, y) \leftarrow_\$ \{1, \ldots, \varphi(N)/2\}; b \leftarrow_\$ \{0,1\} \\ \text{if } b = 0 \text{ then } z \leftarrow_\$ \{1, \ldots, \varphi(N)/2\} \\ \text{if } b = 1 \text{ then } z \coloneqq x \cdot y \ (mod\ \varphi(N)/2) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

2.2 Cryptographic Building Blocks

In the following we introduce the cryptographic primitives used in our work.

Puncturable PseudoRandom Functions. A puncturable pseudorandom function (PRF) is an augmented PRF that has an additional puncturing algorithm. Such an algorithm produces a punctured version of the key that can evaluate the PRF at all points except for the punctured one. It is required that the PRF value at that specific point is pseudorandom even given the punctured key. A puncturable PRF can be constructed from any one-way function [18].

Definition 2 (Puncturable PRFs). *A puncturable family of PRFs is a tuple of polynomial-time algorithms* $(\mathsf{Key}, \mathsf{Puncture}, \mathsf{PRF})$ *defined as follows.*

- $K \leftarrow \mathsf{Key}(1^\lambda)$ *a probabilistic algorithm that takes as input the security parameter and outputs a key K.*
- $K_{-i} \leftarrow \mathsf{Puncture}(K, i)$ *a deterministic algorithm that takes as input a key K and a position $i \in \{0,1\}^n$ and returns a punctured key K_{-i}.*
- $y \leftarrow \mathsf{PRF}(K, i)$ *a deterministic algorithm that takes as input a key K and an index $i \in \{0,1\}^n$ and returns a string $y \in \{0,1\}^m$.*

Definition 3 (Correctness). *For all $\lambda \in \mathbb{N}$, for all outputs $K \leftarrow \mathsf{Key}(1^\lambda)$, for all points $i \in \{0,1\}^n$ and $x \in \{0,1\}^n \setminus i$, and for all $K_{-i} \leftarrow \mathsf{Puncture}(K, i)$, we have that $\mathsf{PRF}(K_{-i}, x) = \mathsf{PRF}(K, x)$.*

Definition 4 (Pseudorandomness). *For all $\lambda \in \mathbb{N}$ and for every polynomial-time adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ there is a negligible function $\mu(\cdot)$, such that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(\tau, K_{-i}, i, y) : \begin{array}{l}(i,\tau) \leftarrow \mathcal{A}_1(1^\lambda) \\ K \leftarrow \mathsf{Key}(1^\lambda) \\ K_{-i} \leftarrow \mathsf{Puncture}(K, i) \\ b \leftarrow_{\$} \{0,1\} \\ \textit{if } b = 0 \textit{ then } y \leftarrow_{\$} \{0,1\}^m \\ \textit{if } b = 1 \textit{ then } y \leftarrow \mathsf{PRF}(K, i)\end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Time-Lock Puzzles. We recall the definition of standard time-lock puzzles [2]. For conceptual simplicity we consider only schemes with binary solutions.

Definition 5 (Time-Lock Puzzles). *A time-lock puzzle is a tuple of two algorithms $(\mathsf{PGen}, \mathsf{PSolve})$ defined as follows.*

- *$Z \leftarrow \mathsf{PGen}(\mathcal{T}, s)$ a probabilistic algorithm that takes as input a hardness-parameter $\mathcal{T}$ and a solution $s \in \{0,1\}$, and outputs a puzzle Z.*
- *$s \leftarrow \mathsf{PSolve}(Z)$ a deterministic algorithm that takes as input a puzzle Z and outputs a solution s.*

Definition 6 (Correctness). *For all $\lambda \in \mathbb{N}$, for all polynomials $\mathcal{T}$ in λ, and for all $s \in \{0,1\}$, it holds that $s = \mathsf{PSolve}(\mathsf{PGen}(\mathcal{T}, s))$.*

Definition 7 (Security). *A scheme $(\mathsf{PGen}, \mathsf{PSolve})$ is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{\mathcal{T}}(\cdot)$ such that for all polynomials $\mathcal{T}(\cdot) \geq \tilde{\mathcal{T}}(\cdot)$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of depth $\leq \mathcal{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[b \leftarrow \mathcal{A}(Z) : Z \leftarrow \mathsf{PGen}(\mathcal{T}(\lambda), b)\right] \leq \frac{1}{2} + \mu(\lambda).$$

Trapdoor Encryption. A trapdoor encryption scheme is a public key encryption scheme that allows one to generate a trapdoor version of the public key. Such trapdoor key is indistinguishable from a normal public key, however encrypting under the trapdoor key hides the message in an information-theoretic sense. Canetti et al. [9] showed that any public-key encryption with perfect re-randomization (such as ElGamal or Paillier encryption) can be used generically to construct such a primitive.

Definition 8 (Trapdoor Encryption). *A trapdoor encryption scheme is a tuple of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{tKeyGen})$ defined as follows.*

- *$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$ a probabilistic algorithm that takes as input the security parameter and outputs a key pair (pk, sk).*
- *$pk \leftarrow \mathsf{tKeyGen}(1^\lambda)$ a probabilistic algorithm that takes as input the security parameter and outputs a trapdoor key pk.*
- *$c \leftarrow \mathsf{Enc}(pk, m)$ a probabilistic algorithm that takes as input a message $m \in \{0,1\}$ and a key pk and returns a ciphertext c.*
- *$m \leftarrow \mathsf{Dec}(sk, c)$ a deterministic algorithm that takes as input a secret key sk and a ciphertext c and returns a message m.*

Definition 9 (Correctness). *For all $\lambda \in \mathbb{N}$, for all $m \in \{0,1\}$ it holds that $m = \mathsf{Dec}(sk, \mathsf{Enc}(pk, m))$, where $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$.*

Definition 10 (Trapdoor Public Keys). *For all $\lambda \in \mathbb{N}$ and for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[b \leftarrow \mathcal{A}(pk) : \begin{array}{l}b \leftarrow_{\$} \{0,1\} \\ \textit{if } b = 0 \textit{ then } pk \leftarrow \mathsf{tKeyGen}(1^\lambda) \\ \textit{if } b = 1 \textit{ then } (pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)\end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Definition 11 (μ-Hiding). *For all $\lambda \in \mathbb{N}$ and for all unbounded adversaries $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[b \leftarrow \mathcal{A}(pk, \mathsf{Enc}(pk, b)) : \begin{array}{l} b \leftarrow_{\$} \{0,1\} \\ pk \leftarrow \mathsf{tKeyGen}(1^\lambda) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

Probabilistic Obfuscation. A probabilistic obfuscator $pi\mathcal{O}$ is an algorithm that obfuscates probabilistic circuits and it can be constructed assuming sub-exponentially secure indistinguishability obfuscation [14] and sub-exponentially secure one-way functions [9].

Definition 12 ($pi\mathcal{O}$ for a class of samplers S). *A uniform polynomial-size machine $pi\mathcal{O}$ is an* indistinguishable obfuscator *for a class of samplers $\mathbf{S}$ over the (possibly randomized) circuit family $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ if, on input a (possibly probabilistic) circuit $C \in \mathcal{C}_\lambda$ and the security parameter 1^λ, outputs a deterministic circuit Λ of size $p(|C|, \lambda)$, for some fixed polynomial $p(\cdot)$.*

Definition 13 (Correctness). *For every non-uniform polynomial-size distinguisher $\mathcal{D}$, every (possibly probabilistic) circuit $C \in \mathcal{C}_\lambda$ and string y, we define the following experiments*

- $\mathsf{EXP}_0^{\mathcal{D}}(1^\lambda, C, y)$*: $\mathcal{D}$ on input $1^\lambda, C, y$, participates in as many number of iterations as he wants. In iteration i, it chooses an input x_i; if $x_i = x_j$ for $j < i$, then abort; else, $\mathcal{D}$ gets back $(C(x_i, r_i))$ where r_i are fresh randomness (r_i = null, if C is deterministic). At the end of the final iteration, $\mathcal{D}$ outputs a bit b. (Note that $\mathcal{D}$ is stateful.)*
- $\mathsf{EXP}_1^{\mathcal{D}}(1^\lambda, C, y)$*: Obfuscate circuit C to obtain $\Lambda \leftarrow pi\mathcal{O}(1^\lambda, C; r)$ using fresh randomness r. Run $\mathcal{D}$ as described in the above experiment, except that in each iteration give $\Lambda(x_i)$ to $\mathcal{D}$ instead.*

We require that for every non-uniform polynomial-size distinguisher $\mathcal{D}$, there is a negligible function $\mu(\cdot)$, such that, for every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, and every polynomial-size auxiliary input y it holds that

$$\Pr[b \leftarrow \mathsf{EXP}_b^{\mathcal{D}}(1^\lambda, C, y)] \leq \frac{1}{2} + \mu(\lambda).$$

Definition 14 (Security with respect to S). *For every sampler $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathbf{S}$, and for every non-uniform polynomial-size machine $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr[b \leftarrow \mathcal{A}(C_0, C_1, pi\mathcal{O}(1^\lambda, C_b), y) : b \leftarrow_{\$} \{0,1\}; (C_0, C_1, y) \leftarrow D_\lambda)].$$

Indistinguishability Obfuscation. We can cast the definition of indistinguishability obfuscation (iO) for circuits as a special case of worst-case input $pi\mathcal{O}$ for the class $\mathcal{C}' = \{\mathcal{C}'_\lambda\}_{\lambda\in\mathbb{N}}$ of deterministic circuits as done in [9].

Definition 15 (iO for Circuits [14]). *A uniform PPT machine $i\mathcal{O}$ is an indistinguishable obfuscator for circuits, if it is a $pi\mathcal{O}$ for the class of worst-case input Indistinguishability samplers $\mathbf{S}^{\mathsf{w-Ind}}$ over $\mathcal{C}'$ that includes all* deterministic *circuits of size at most λ.*

What is left to be defined is the class of worst-case input samplers.

Definition 16 (Worst-case input Indistinguishable Samplers). *The class $\mathbf{S}^{\mathsf{w-Ind}}$ of worst-case input indistinguishable samplers for a circuit family $\mathcal{C}$ contains all circuit samplers $D = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ for $\mathcal{C}$ with the following property: For all adversary $\mathcal{A} = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathcal{A}_1$ is an unbounded non-uniform machine and $\mathcal{A}_2$ is PPT, there is a negligible function $\mu(\cdot)$, such that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(st, C_0, C_1, z, x, y) : \begin{array}{l} (C_0, C_1, z) \leftarrow D_\lambda \\ (x, st) \leftarrow \mathcal{A}_1(C_0, C_1, z) \\ b \leftarrow_{\$} \{0,1\} \\ y \leftarrow C_b(x) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

3 Homomorphic Time-Lock Puzzles

In the following we give a definition for the main object of interest of this work, homomorphic time-lock puzzles (HTLP). The syntax follows the standard notation for time-lock puzzles except that we consider an additional setup phase that depends on the hardness parameter but not on the secret. Furthermore, HTLPs are augmented with an evaluation algorithm that allows one to manipulate puzzles in a meaningful way.

Definition 17 (Homomorphic Time-Lock Puzzles). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of circuits and let $\mathcal{S}$ be a finite domain. A homomorphic time-lock puzzle (HTLP) with respect to $\mathcal{C}$ and with solution space $\mathcal{S}$ is tuple of four algorithms* $(\mathsf{HP.PSetup}, \mathsf{HP.PGen}, \mathsf{HP.PSolve}, \mathsf{HP.PEval})$ *defined as follows.*

- $pp \leftarrow \mathsf{HP.PSetup}(1^\lambda, \mathcal{T})$ *a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter $\mathcal{T}$, and outputs public parameters pp.*
- $Z \leftarrow \mathsf{HP.PGen}(pp, s)$ *a probabilistic algorithm that takes as input public parameters pp, and a solution $s \in \mathcal{S}$, and outputs a puzzle Z.*
- $s \leftarrow \mathsf{HP.PSolve}(pp, Z)$ *a deterministic algorithm that takes as input public parameters pp and a puzzle Z and outputs a solution s.*
- $Z' \leftarrow \mathsf{HP.PEval}(C, pp, Z_1, \ldots, Z_n)$ *a probabilistic algorithm that takes as input a circuit $C \in \mathcal{C}_\lambda$, public parameters pp and a set of n puzzles $(Z_1, \ldots, Z_n)$ and outputs a puzzle Z'.*

Security requires that the solution of the puzzles is hidden for all adversaries that run in (parallel) time less than $\mathcal{T}$. Here we consider a basic version where the time is counted from the moment the public parameters are published. We also consider a stronger version, i.e., where the time is taken from the moment each puzzle is generated, in Section 5.2.

Definition 18 (Security of HTLP). *An HTLP scheme* $(\mathsf{HP.PSetup}, \mathsf{HP.PGen}, \mathsf{HP.PSolve}, \mathsf{HP.PEval})$ *is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{\mathcal{T}}(\cdot)$ such that for all polynomials $\mathcal{T}(\cdot) \geq \tilde{\mathcal{T}}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $\mathcal{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(pp, Z, \tau) : \begin{array}{l}(\tau, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda) \\ pp \leftarrow \mathsf{HP.PSetup}(1^\lambda, \mathcal{T}(\lambda)) \\ b \leftarrow_{\$} \{0,1\} \\ Z \leftarrow \mathsf{HP.PGen}(pp, s_b)\end{array}\right] \leq \frac{1}{2} + \mu(\lambda)$$

and $(s_0, s_1) \in \mathcal{S}^2$.

We consider the basic notion of correctness, that concerns with a single application of the evaluation algorithm. The definition can be easily extended to the multi-hop settings (in the same spirit as [17]) in a natural way.

Definition 19 (Correctness). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of circuits (together with their respective representations). An HTLP scheme* $(\mathsf{HP.PSetup}, \mathsf{HP.PGen}, \mathsf{HP.PSolve}, \mathsf{HP.PEval})$ *is correct (for the class $\mathcal{C}$) if for all $\lambda \in \mathbb{N}$, all polynomials $\mathcal{T}$ in λ, all circuits $C \in \mathcal{C}_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in \mathcal{S}^n$, all pp in the support of $\mathsf{HP.PSetup}(1^\lambda, \mathcal{T})$, and all Z_i in the support of $\mathsf{HP.PGen}(pp, s_i)$, the following two conditions are satisfied:*

- *There exists a negligible function $\mu(\cdot)$ such that*

$$\Pr\left[\mathsf{HP.PSolve}(pp, \mathsf{HP.PEval}(C, pp, Z_1, \ldots, Z_n)) \neq C(s_1, \ldots, s_n)\right] \leq \mu(\lambda).$$

- *There exists a fixed polynomial $p(\cdot)$ such that the runtime of $\mathsf{HP.PSolve}(pp, Z)$ is bounded by $p(\lambda, \mathcal{T})$, where $Z \leftarrow \mathsf{HP.PEval}(C, pp, Z_1, \ldots, Z_n)$.*

The central property for HTLPs is compactness, which requires that the size of evaluated ciphertexts is independent of the size of the circuit and that the running time of the evaluation algorithm is independent of the hardness parameter.

Definition 20 (Compactness). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of circuits (together with their respective representations). An HTLP scheme* $(\mathsf{HP.PSetup}, \mathsf{HP.PGen}, \mathsf{HP.PSolve}, \mathsf{HP.PEval})$ *is compact (for the class $\mathcal{C}$) if for all $\lambda \in \mathbb{N}$, all polynomials $\mathcal{T}$ in λ, all circuits $C \in \mathcal{C}_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in \mathcal{S}^n$, all pp in the support of $\mathsf{HP.PSetup}(1^\lambda, \mathcal{T})$, and all Z_i in the support of $\mathsf{HP.PGen}(pp, s_i)$, the following two conditions are satisfied:*

- *There exists a fixed polynomial $p(\cdot)$ such that $|Z| = p(\lambda, |C(s_1, \ldots, s_n)|)$, where $Z \leftarrow \mathsf{HP.PEval}(C, pp, Z_1, \ldots, Z_n)$.*
- *There exists a fixed polynomial $\tilde{p}(\cdot)$ such that the runtime of $\mathsf{HP.PEval}(C, pp, Z_1, \ldots, Z_n)$ is bounded by $\tilde{p}(\lambda, |C|)$.*

Finally we observe that one can define circuit privacy for HTLPs analogously to the FHE notion. Since it is not of significance for our applications we refrain from giving a formal definition and we refer the reader to [27].

4 Constructions

In this section we describe our HTLP schemes for different classes of functions.

4.1 Linearly Homomorphic

We describe a scheme (LHTLP) homomorphic over the ring $(\mathbb{Z}_N, +)$ below.

$\mathsf{LHP.PSetup}(1^\lambda, \mathcal{T})$:

- Sample a pair of primes (p, q) such that $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also primes, and set $N \coloneqq p \cdot q$.
- Sample a uniform $\tilde{g} \leftarrow_\$ \mathbb{Z}_N^*$ and set $g \coloneqq -\tilde{g}^2 \pmod N$.
- Compute $h \coloneqq g^{2^\mathcal{T}}$, which can be optimized by reducing $2^\mathcal{T}$ modulo $\varphi(N)/2$ first.
- Output $pp \coloneqq (\mathcal{T}, N, g, h)$.

$\mathsf{LHP.PGen}(pp, s)$:

- Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
- Sample a uniform $r \leftarrow_\$ \{1, \ldots, N^2\}$.
- Generate the elements $u \coloneqq g^r \pmod N$ and $v \coloneqq h^{r\cdot N} \cdot (1 + N)^s \pmod{N^2}$.
- Output $Z \coloneqq (u, v)$ as the puzzle.

$\mathsf{LHP.PSolve}(pp, Z)$:

- Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
- Parse the puzzle $Z \coloneqq (u, v)$.
- Compute $w \coloneqq u^{2^\mathcal{T}} \pmod N$ by repeated squaring.
- Output $s \coloneqq \frac{v/(w)^N \pmod{N^2}-1}{N}$ as the solution.

$\mathsf{LHP.PEval}(\oplus, pp, Z_1, \ldots, Z_n)$:

- Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
- Parse every $Z_i \coloneqq (u_i, v_i) \in \mathbb{J}_N \times \mathbb{Z}_{N^2}^*$.

– Compute $\tilde{u} \coloneqq \prod_{i=1}^{n} u_i \pmod{N}$ and $\tilde{v} \coloneqq \prod_{i=1}^{n} v_i \pmod{N^2}$.
– Output the puzzle $(\tilde{u}, \tilde{v})$.

To see why the scheme is correct, observe that

$$\begin{aligned}
\tilde{s} &= \frac{\tilde{v}/(\tilde{w})^N \pmod{N^2} - 1}{N} \\
&= \frac{\prod_{i=1}^{n} v_i / \left(\prod_{i=1}^{n} u_i^{2^\mathcal{T}} \pmod{N}\right)^N \pmod{N^2} - 1}{N} \\
&= \frac{\prod_{i=1}^{n} h^{r_i \cdot N} \cdot (1+N)^{s_i} / \left(\prod_{i=1}^{n} h^{r_i} \pmod{N}\right)^N \pmod{N^2} - 1}{N} \\
&= \frac{\prod_{i=1}^{n} h^{r_i \cdot N} \cdot (1+N)^{s_i} / \prod_{i=1}^{n} h^{r_i \cdot N} \pmod{N^2} - 1}{N} \\
&= \frac{(1+N)^{\sum_{i=1}^{n} s_i} \pmod{N^2} - 1}{N}
\end{aligned}$$

by Lemma 2. Furthermore,

$$\tilde{s} = \frac{(1+N)^{\sum_{i=1}^{n} s_i} \pmod{N^2} - 1}{N} = \frac{1 + N \cdot \sum_{i=1}^{n} s_i - 1}{N} = \sum_{i=1}^{n} s_i$$

by binomial expansion. The security of our construction is shown in the following.

Theorem 1. *Let N be a strong RSA integer. If the sequential squaring assumption and the DDH assumptions hold over $\mathbb{J}_N$ and the DCR assumption hold over $Z^*_{N^2}$, then the scheme* LHTLP *is a secure homomorphic time-lock puzzle.*

Proof. Consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0$: Is defined as the original scheme.

Hybrid $\mathcal{H}_1$: In this hybrid h is sampled uniformly from $\mathbb{J}_N$, instead of being computed as $h \coloneqq g^{2^\mathcal{T}}$. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an efficient distinguisher where the depth of $\mathcal{A}_2$ is less than $\mathcal{T}$. We construct the following reduction against the sequential squaring assumption: The reduction runs the adversary $\mathcal{A}_1$ on input the security parameter 1^λ and receives two secrets (s_0, s_1) and some advice τ. Then receives as input the tuple $(N, g, \mathcal{T}, x, y)$, sets $pp \coloneqq (\mathcal{T}, N, x, y)$ and computes Z exactly as specified by the scheme using s_b as the solution, for a random $b \leftarrow_\$ \{0, 1\}$. Then it invokes the adversary $\mathcal{A}_2$ on input (pp, Z, τ) and outputs whatever $\mathcal{A}_2$ returns. Observe that the depth of the reduction is only a constant fraction larger than that of $\mathcal{A}_2$. We the analyze the two cases separately.

1. (N, g, x, y) is a uniform tuple: Then $x = g$ and $y = h$ are uniform in $\mathbb{J}_N$. Thus
$$(\mathcal{T}, N, x, y) = (\mathcal{T}, N, g, h)$$
is distributed as in $\mathcal{H}_1$.
2. (N, g, x, y, z) is a squared tuple: In this case we have that $(N, g, x, y) = (N, g, x, x^{2^\mathcal{T}})$. Which means that the tuple
$$(\mathcal{T}, N, x, y) = (\mathcal{T}, N, g, g^{2^\mathcal{T}})$$
is distributed according to $\mathcal{H}_0$.

Thus the existence of an efficient distinguisher (with depth smaller than $\mathcal{T}$) between the two hybrids contradicts the sequential squaring assumption.

Hybrid $\mathcal{H}_2$: In this hybrid r is sampled from the set $\{1, \ldots, \varphi(N)/2\}$, rather than $\{1, \ldots, N^2\}$. The two hybrids are statistically indistinguishable by Lemma 1. We stress that the encrypter does not know $\varphi(N)/2$, however the argument is purely statistical and therefore there is no need for a polynomial-time simulation.

Hybrid $\mathcal{H}_3$: In this hybrid u is sampled uniformly at random from $\mathbb{J}_N$. We show indistinguishability with a reduction against the DDH assumption over $\mathbb{J}_N$. The reduction runs the adversary on input the security parameter to receive (τ, s_0, s_1). On input (N, g, g^x, g^y, g^z), the reduction sets the public parameters of the scheme to $(\mathcal{T}, N, g, g^x)$ and the puzzle to $(g^y, (g^z)^N \cdot (1+N)^{s_b})$, then feeds $\mathcal{A}_2$ with those inputs and it returns whatever the adversary returns. Clearly the reduction is efficient, so what is left to be shown is that the inputs are distributed correctly, according to the two hybrids.

1. (N, g, g^x, g^y, g^z) is a uniform tuple: Then $g^x = h$ is uniform in $\mathbb{J}_N$, $g^y = u$ is uniform in $\mathbb{J}_N$, and g^z is uniform in $\mathbb{J}_N$, so we rewrite it as $g^z = h^r$ (for some random $r \in \{1, \ldots, \varphi(N)/2\}$). Thus
$$(\mathcal{T}, N, g, g^x), (g^y, (g^z)^N \cdot (1+N)^{s_b}) = (\mathcal{T}, N, g, h), (u, h^{r \cdot N} \cdot (1+N)^{s_b})$$
are distributed identically to $\mathcal{H}_3$.
2. (N, g, g^x, g^y, g^z) is a DDH tuple: For the sake of clarity we rewrite the input tuple as (N, g, g^x, g^y, g^{xy}). Fix $g^x = h$ and observe that the tuples
$$(\mathcal{T}, N, g, g^x), (g^y, (g^{xy})^N \cdot (1+N)^{s_b}) = (\mathcal{T}, N, g, h), (g^y, h^{y \cdot N} \cdot (1+N)^{s_b})$$
are distributed according to $\mathcal{H}_2$.

It follows that any non negligible advantage in distinguishing the two hybrids directly implies an attack against DDH.

Hybrid $\mathcal{H}_4$: In this hybrid v is computed as $w \cdot (1+N)^{s_b} \pmod{N^2}$, where w is uniformly sampled from $\mathbb{Z}^*_{N^2}$ (constrained on having Jacobi symbol $+1$). Consider the following reduction against the DCR assumption: Prior to the challenge, the reduction runs $\mathcal{A}_1$ on input 1^λ and receives (τ, s_0, s_1). On input (N, y), the reduction sets N as the modulus and samples g and h uniformly from $\mathbb{J}_N$ (as specified in the $\mathcal{H}_3$). Then it computes the Jacobi symbol of y and samples some $\tilde{y}$ with the same Jacobi symbol as y. Then it samples some $u \leftarrow_\$ \mathbb{J}_N$ and sets $v \coloneqq y \cdot \tilde{y}^N \cdot (1+N)^{s_b} \pmod{N^2}$, for a uniform $b \leftarrow_\$ \{0,1\}$. Finally it runs $\mathcal{A}_2$ on input $((\mathcal{T}, N, g, h), (u, v))$ and returns whatever $\mathcal{A}_2$ returns. Note that the reduction is efficient since the Jacobi symbol is efficiently computable without the factorization of N. If y is uniform in $\mathbb{Z}^*_{N^2}$, then so is $y \cdot \tilde{y}^N \pmod{N^2}$, and therefore the reduction perfectly simulates $\mathcal{H}_4$. On the other hand if y is an N-th residue, then $y \cdot \tilde{y}^N = x^N \cdot \tilde{y}^N = (x\tilde{y})^N \pmod{N^2}$ is also an N-th residue. Note that the Jacobi symbol of $x\tilde{y}$ is $+1$, since the Jacobi symbol is multiplicatively homomorphic. It follows that in this case the inputs of the reduction are identical to that of $\mathcal{H}_3$. We can therefore bound from above the distance between these two hybrids by a negligible amount.

Observe that in the last hybrid every bit of information about the message is lost. This concludes our proof.

4.2 Multiplicatively Homomorphic

In the following we describe our scheme (MHTLP) which is multiplicatively homomorphic over the ring $(\mathbb{J}_N, \cdot)$. The algorithms are described below.

MHP.PSetup$(1^\lambda, \mathcal{T})$:

- Sample a pair of primes (p, q) such that $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also primes, and set $N \coloneqq p \cdot q$.
- Sample a uniform $\tilde{g} \leftarrow_\$ \mathbb{Z}^*_N$ and set $g \coloneqq -\tilde{g}^2 \pmod{N}$.
- Compute $h \coloneqq g^{2^\mathcal{T}}$, which can be optimized by reducing $2^\mathcal{T}$ modulo $\varphi(N)/2$ first.
- Output $pp \coloneqq (\mathcal{T}, N, g, h)$.

MHP.PGen(pp, s) :

– Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
– Sample a uniform $r \leftarrow_{\$} \{1, \ldots, N^2\}$.
– Generate the elements $u \coloneqq g^r \pmod N$ and $v \coloneqq h^r \cdot s \pmod N$.
– Output $Z \coloneqq (u, v)$ as the puzzle.

$\mathsf{MHP.PSolve}(pp, Z)$:

– Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
– Parse the puzzle $Z \coloneqq (u, v)$.
– Compute $w \coloneqq u^{2^{\mathcal{T}}} \pmod N$ by repeated squaring.
– Output $s \coloneqq v/w$ as the solution.

$\mathsf{MHP.PEval}(\otimes, pp, Z_1, \ldots, Z_n)$:

– Parse $pp \coloneqq (\mathcal{T}, N, g, h)$.
– Parse every $Z_i \coloneqq (u_i, v_i) \in \mathbb{J}_N^2$.
– Compute $\tilde{u} \coloneqq \prod_{i=1}^n u_i \pmod N$ and $\tilde{v} \coloneqq \prod_{i=1}^n v_i \pmod N$.
– Output the puzzle $(\tilde{u}, \tilde{v})$.

For correctness it suffices to observe that

$$\tilde{s} = \frac{\tilde{v}}{\tilde{w}} = \frac{\tilde{v}}{\tilde{u}^{2^{\mathcal{T}}}} = \frac{\prod_{i=1}^n v_i}{\prod_{i=1}^n u_i^{2^{\mathcal{T}}}} = \frac{\prod_{i=1}^n h^{r_i} \cdot s_i}{\prod_{i=1}^n g^{r_i \cdot 2^{\mathcal{T}}}} = \frac{\prod_{i=1}^n h^{r_i} \cdot s_i}{\prod_{i=1}^n h^{r_i}} = \prod_{i=1}^n s_i.$$

For security we prove the following theorem.

Theorem 2. *Let N be a strong RSA integer. If the sequential squaring and the DDH assumptions hold over $\mathbb{J}_N$, then the scheme* MHTLP *is a secure homomorphic time-lock puzzle.*

Proof. Consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0$: Is defined as the original scheme.

Hybrid $\mathcal{H}_1$: Same as Theorem 1.

Hybrid $\mathcal{H}_2$: Same as Theorem 1.

Hybrid $\mathcal{H}_3$: In this hybrid v is computed as $w \cdot s$, for a uniform $w \leftarrow_{\$} \mathbb{J}_N$. Indistinguishability follows from an invocation of the DDH assumption over $\mathbb{J}_N$: The reduction runs the adversary on input the security parameter and receives (τ, s_0, s_1). On input (N, g, g^x, g^y, g^z), the reduction sets the public parameters of the scheme to $(\mathcal{T}, N, g, g^x)$ and computes the puzzle Z as $(g^y, g^z \cdot s_b)$, for a randomly sampled $b \leftarrow_{\$} \{0, 1\}$. The adversary is fed with (pp, Z, τ) and the reduction returns whatever the adversary returns. The reduction is clearly polynomial-time. We consider the two distributions in the following.

1. (N, g, g^x, g^y, g^z) is a uniform tuple: Then the tuples
$$(\mathcal{T}, N, g, g^x), (g^y, g^z \cdot s_b) = (\mathcal{T}, N, g, h), (g^y, w \cdot s_b)$$
are distributed identically to $\mathcal{H}_3$.
2. (N, g, g^x, g^y, g^z) is a DDH tuple: For the sake of clarity we rewrite the input tuple as (N, g, g^x, g^y, g^{xy}). Fix $g^x = h$ and observe that the tuples
$$(\mathcal{T}, N, g, g^x), (g^y, g^{xy} \cdot s_b) = (\mathcal{T}, N, g, h), (g^y, h^y \cdot s_b)$$
are distributed according to $\mathcal{H}_2$.

It follows that any non negligible advantage in distinguishing the two hybrids directly implies an attack against DDH.

The proof is concluded by observing that in $\mathcal{H}_3$ the secret s_b is information-theoretically hidden by w.

XOR-Homomorphism. If we set N to be a Blum integer and encode the secret $s \in \{0,1\}$ as $(-1)^s$, then the same construction gives us an XOR homomorphic scheme. This is because if N is a Blum integer, then $(\pm 1, \cdot)$ is a subgroup of $\mathbb{J}_N$.

4.3 Fully Homomorphic

In the following we describe our construction for a fully-homomorphic time-lock puzzle (FHTLP). Without loss of generality we consider binary secrets and circuits that are composed exclusively by NAND gates. Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{tKeyGen})$ be a trapdoor encryption scheme, $(\mathsf{Key}, \mathsf{Puncture}, \mathsf{PRF})$ be a puncturable PRF, $(\mathsf{PGen}, \mathsf{PSolve})$ be any (non-homomorphic) time-lock puzzle, $pi\mathcal{O}$ be an obfuscator for probabilistic circuits, and $i\mathcal{O}$ be an obfuscator for deterministic circuits. Define the circuit $\mathsf{Prog}^{(sk,pk)}(\alpha,\beta)$ and $\mathsf{MProg}^{(sk_0,k,k')}(i)$ as

$\mathsf{Prog}^{(sk,pk)}(\alpha,\beta)$:

parse $\alpha \coloneqq (z_\alpha, c_\alpha)$
parse $\beta \coloneqq (z_\beta, c_\beta)$
$s_\alpha \leftarrow \mathsf{Dec}(sk, c_\alpha), s_\beta \leftarrow \mathsf{Dec}(sk, c_\beta)$
$s \coloneqq s_\alpha$ NAND s_β
$z \leftarrow \mathsf{PGen}(\mathcal{T}, s)$
$c \leftarrow \mathsf{Enc}(pk, s)$
return (z, c)

$\mathsf{MProg}^{(sk_0,k,k')}(i)$:

$r_{i-1} \leftarrow \mathsf{PRF}(k, i-1)$
$r_i \leftarrow \mathsf{PRF}(k, i), r'_i \leftarrow \mathsf{PRF}(k', i)$
$(pk_{i-1}, sk_{i-1}) \leftarrow \mathsf{KeyGen}(1^\lambda; r_{i-1})$
$(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda, r_i)$
$P_i \leftarrow \mathsf{Prog}^{(sk_{i-1}, pk_i)}$
$\Lambda_i \leftarrow pi\mathcal{O}(1^p, P_i; r'_i)$
return (Λ_i)

Let $\mathcal{L}$ be a super-polynomial function $\mathcal{L}(\lambda) \coloneqq 2^{\omega(\log(\lambda))}$. The four algorithms of the scheme are described below.

$\mathsf{FHP.PSetup}(1^\lambda, \mathcal{T})$:

- Sample a pair of keys $(pk_0, sk_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- Sample two PRF keys $k, k' \leftarrow \mathsf{Key}(1^\lambda)$
- Obfuscate using $i\mathcal{O}$ the circuit $\mathsf{MProg}^{(sk_0,k,k')}$, that is, sample $MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{MProg}^{(sk_0,k,k')})$ where the security parameter $p = p(\lambda)$ for obfuscation is an upper-bound on the size of $\mathsf{MProg}^{(sk_0,k,k')}$.
- Output $pp \coloneqq (\mathcal{T}, pk_0, MEvk)$.

$\mathsf{FHP.PGen}(pp, s)$:

- Parse $pp \coloneqq (\mathcal{T}, pk_0, MEvk)$.
- Generate a ciphertext $c \leftarrow \mathsf{Enc}(pk_0, s)$.
- Generate a puzzle $z \leftarrow \mathsf{PGen}(\mathcal{T}, s)$.
- Output $Z \coloneqq (z, c)$ as the puzzle.

$\mathsf{FHP.PSolve}(pp, Z)$:

- Parse the puzzle $Z \coloneqq (z, c)$.
- Compute $s \leftarrow \mathsf{PSolve}(z)$ and output s as the solution.

$\mathsf{FHP.PEval}(C, pp, Z_1, \ldots, Z_n)$:

- Evaluate C (of depth $\ell \leq \mathcal{L}(\lambda)$) layer by layer. For iteration $i \in \{0, \ldots, \ell\}$, generate the evaluation key for the layer as $\Lambda_i \leftarrow MEvk(i)$.
- For each NAND gate g in this layer i, let $\alpha(g), \beta(g)$ be the puzzles of the values of its input wires

– Evaluate g homomorphically by computing $\gamma(g) = \Lambda_i(\alpha(g), \beta(g))$ as the puzzle of the value of g's output wire.
– Output the puzzle generated in the last iteration ℓ.

Correctness easily follows from the correctness of the underlying primitives. Towards arguing about security, we define a useful subroutine $\mathsf{tProg}^{(tpk)}(\alpha, \beta)$ as follows

$\underline{\mathsf{tProg}^{(tpk)}(\alpha, \beta):}$
$z \leftarrow \mathsf{PGen}(\mathcal{T}, 0)$
$c \leftarrow \mathsf{Enc}(tpk, 0)$
return (z, c)

which is instrumental for probabilistic obfuscator $pi\mathcal{O}$. Let $SK = \{sk_\lambda\}$ be the set of all strings of length $n = n(\lambda)$. Define the distribution D^{SK} as follows: Sample a trapdoor key $tpk \leftarrow \mathsf{tKeyGen}(1^\lambda)$ and some $sk \leftarrow_\$ SK$ and return $(C_0 = \mathsf{Prog}^{(sk,tpk)}, C_1 = \mathsf{tProg}^{(tpk)}, tpk)$. Then $\mathbf{S}$ is the class of samplers that include the distribution ensembles D^{SK} for all strings SK of length n. Security is established by the following theorem and the proof is given in Section A.

Theorem 3. *Let* $(\mathsf{PGen}, \mathsf{PSolve})$ *be a secure time-lock puzzle. Define* $\mu(\lambda) \coloneqq \tilde{\mu}(\lambda) \cdot \mathcal{L}^{-1}$*, where* $\tilde{\mu}(\cdot)$ *is some negligible function. Assume the following primitives with distinguishing gaps bounded by* $\mu(\lambda)$ *against a polynomial-size adversary who's depth is bounded by* $\mathcal{T}^\varepsilon(\lambda)$*, for some constant* $\varepsilon < 1$*:*

– $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{tKeyGen})$ *is a secure* μ*-hiding trapdoor encryption scheme,*
– $pi\mathcal{O}$ *is a secure indistinguishable obfuscator for the class of samplers* $\mathbf{S}$*,*
– $i\mathcal{O}$ *is a secure indistinguishable obfuscator for circuits, and*
– $(\mathsf{Key}, \mathsf{Puncture}, \mathsf{PRF})$ *is a secure puncturable PRF.*

Then, the scheme FHTLP *is a secure homomorphic time-lock puzzle.*

5 Extensions

In the following we explore and discuss several extensions of our constructions.

5.1 Semi-Compact Scheme for Branching Programs

The linearly homomorphic scheme described in Section 4.1 can be easily generalized to higher powers of N, along the lines of the work of Damgrd and Jurik [11], where the message domain is $\mathbb{Z}_{N^{y-1}}$ and the ciphertexts live in $\mathbb{Z}_{N^y}$, for an arbitrary $y \in \mathbb{N}$. The public parameters are identical to the ones generated by LHP.PSetup, whereas the puzzle is generated as

$$u \coloneqq g^r \pmod{N} \text{ and } v \coloneqq h^{r \cdot N^{y-1}} \cdot (1+N)^s \pmod{N^y}.$$

The solving algorithm factors $h^{r \cdot N^{y-1}} \pmod{N^y}$ out of v, via a series of sequential squarings, and recovers s from $(1+N)^s \pmod{N^y}$ using the polynomial-time discrete-logarithm algorithm described in [11]. Security follows from a natural generalization of the DCR assumption, also introduced in [11].

Note that the asymptotic message-ciphertext rate approaches 1 as y grows. This is desirable from a practical perspective but also it allows us to instantiate the compiler of Ishai and Paskin [21] with our extended scheme: As a corollary we obtain a (semi-compact) HTLP for branching programs (a superclass of NC1) where the ciphertext size grows linearly in the length of the branching program but does not depend on its width.

5.2 Reusing the Setup

A shortcoming of our primitive is that security is guaranteed to hold against a depth-constrained adversary that takes as input both the public parameters pp and the puzzle Z. This is equivalent to saying that the secrets are hidden until time $\mathcal{T}$ since the generation of the setup rather than the generation of the puzzle. From a practical perspective, this cripples the applicability of our primitive since the public parameters need to be re-initialized after time $\mathcal{T}$.

Ideally, we would like to set the public parameters once and for all and compute polynomially many puzzles at arbitrary time intervals. Each puzzle should then hide the secret until time $\mathcal{T}$, starting from the generation of the puzzle itself. Thus we consider a two stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ is polynomial-size (unbounded depth) and is allowed to craft the polynomial-size advice τ *after* being given the the public parameters pp. Then the depth-bounded $\mathcal{A}_2$ is asked to guess the bit b on input the puzzle Z and the advice τ. This is formalized in the following.

Definition 21 (Reusable Security of HTLP). *An HTLP scheme* $(\mathsf{HP.PSetup}, \mathsf{HP.PGen}, \mathsf{HP.PSolve}, \mathsf{HP.PEval})$ *is reusable secure with gap* $\varepsilon < 1$ *if there exists a polynomial* $\tilde{\mathcal{T}}(\cdot)$ *such that for all polynomials* $\mathcal{T}(\cdot) \geq \tilde{\mathcal{T}}(\cdot)$ *and every polynomial-size adversary* $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ *where the depth of* $\mathcal{A}_2$ *is bounded from above by* $\mathcal{T}^\varepsilon(\lambda)$*, there exists a negligible function* $\mu(\cdot)$*, such that for all* $\lambda \in \mathbb{N}$ *it holds that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(Z, \tau) : \begin{array}{l} pp \leftarrow \mathsf{HP.PSetup}(1^\lambda, \mathcal{T}(\lambda)) \\ (\tau, s_0, s_1) \leftarrow \mathcal{A}_1(pp) \\ b \leftarrow_{\$} \{0,1\} \\ Z \leftarrow \mathsf{HP.PGen}(pp, s_b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda)$$

and $(s_0, s_1) \in \mathcal{S}^2$.

Arguing about the security of the constructions described in Section 4.1 and Section 4.2 in these settings requires a slightly modified version of the standard sequential squaring assumption (Assumption 1) that we describe below.

Assumption 4 (Strong Sequential Squaring) *Let* N *be a uniformly sampled strong RSA integer,* g *be a generator of* $\mathbb{J}_N$*, and* $\mathcal{T}(\cdot)$ *be a polynomial. Then there exists some* $0 < \varepsilon < 1$ *such that for every polynomial-size adversary* $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$*, where the depth of* $\mathcal{A}_2$ *is bounded from above by* $\mathcal{T}^\varepsilon(\lambda)$*, there exists a negligible function* $\mu(\cdot)$ *such that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(x, y, \tau) : \begin{array}{l} \tau \leftarrow \mathcal{A}_1(N, g, \mathcal{T}(\lambda)) \\ x \leftarrow_{\$} \mathbb{J}_N; b \leftarrow_{\$} \{0,1\} \\ \textit{if } b = 0 \textit{ then } y \leftarrow_{\$} \mathbb{J}_N \\ \textit{if } b = 1 \textit{ then } y \coloneqq x^{2^{\mathcal{T}(\lambda)}} \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

This essentially corresponds to stating that the prior knowledge of the group structure does not help one breaking the sequentiality of the squaring operation, which seems to be a mild strengthening of the original conjecture. We remark that similar assumptions have already appeared in the context of verifiable delay functions [29,31,5]. We are now ready to state the following theorems.

Theorem 4. *Let* N *be a strong RSA integer. If the strong sequential squaring assumption and the DCR assumption hold over* $\mathbb{J}_N$ *and* $Z^*_{N^2}$*, respectively, then the scheme* LHTLP *is a reusable secure homomorphic time-lock puzzle.*

Proof. Consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0$**:** This is the original experiment.

Hybrid $\mathcal{H}_1$**:** In this hybrid r is randomly sampled from $\{1, \ldots, \varphi(N)/2\}$. By Lemma 1, $\mathcal{H}_0$ and $\mathcal{H}_1$ are statistically close.

Hybrid $\mathcal{H}_2$: In this hybrid v is computed as $z^N \cdot (1+N)^{s_b} \pmod{N^2}$, for a uniform $z \leftarrow_\$ \mathbb{J}_N$. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an efficient distinguisher where the depth of $\mathcal{A}_2$ is less than $\mathcal{T}$. We construct the following reduction $(\mathcal{R}_1, \mathcal{R}_2)$ against the strong sequential squaring assumption: $\mathcal{R}_1$ takes as input the tuple $(N, g, \mathcal{T})$ and computes $h \coloneqq g^{2^\mathcal{T}}$, then it sets $pp \coloneqq (\mathcal{T}, N, g, h)$ and runs $\mathcal{A}_1(pp)$, who outputs some (τ, s_0, s_1), which is also the output of $\mathcal{R}_1$. The challenger sends to $\mathcal{R}_2$ the triple $(x, y, (\tau, s_0, s_1))$, who sets $u \coloneqq x$ and $v \coloneqq y^N \cdot (1+N)^{s_b} \pmod{N^2}$, for a random $b \leftarrow_\$ \{0,1\}$, and runs $\mathcal{A}_2((u,v), \tau)$ outputting whatever the adversary outputs. Observe that $\mathcal{R}_1$ is efficient, since $\mathcal{T}$ is a polynomial, and that the depth of $\mathcal{R}_2$ is identical (up to a constant factor) to that of $\mathcal{A}_2$. We distinguish two cases.

1. $y = x^{2^\mathcal{T}}$: Let $x = g^r$, for some $r \in \{1, \ldots, \varphi(N)/2\}$. Then the puzzle
$$(u, v) = (x, x^{2^\mathcal{T} \cdot N} \cdot (1+N)^{s_b} \pmod{N^2}) = (g^r, h^{r \cdot N} \cdot (1+N)^{s_b} \pmod{N^2})$$
is distributed according to $\mathcal{H}_1$.
2. $y \leftarrow_\$ \mathbb{J}_N$: In this case the puzzle
$$(u, v) = (x, y^N \cdot (1+N)^{s_b} \pmod{N^2})$$
is distributed according to $\mathcal{H}_2$.

Thus the existence of $(\mathcal{R}_1, \mathcal{R}_2)$ contradicts the sequential squaring assumption.

Hybrid $\mathcal{H}_3$: In this hybrid v is computed as $w \cdot (1+N)^{s_b} \pmod{N^2}$, where w is uniformly sampled from $\mathbb{Z}^*_{N^2}$ (constrained to have Jacobi symbol $+1$). The indistinguishability follows from an invocation of the DCR assumption and the argument is identical to the last hybrid of Theorem 1.

The proof concludes by observing that the message in the last hybrid is hidden in an information-theoretic sense.

Theorem 5. *Let N be a strong RSA integer. If the strong sequential squaring assumption holds over $\mathbb{J}_N$, then the scheme* MHTLP *is a secure reusable homomorphic time-lock puzzle.*

Proof. Consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0$: This is the original experiment.

Hybrid $\mathcal{H}_1$: Same as Theorem 4.

Hybrid $\mathcal{H}_2$: In this hybrid we compute v as $w \cdot s$, for a uniform $w \leftarrow_\$ \mathbb{J}_N$. The two hybrids are indistinguishable by the sequential squaring assumption over $\mathbb{J}_N$. Consider the following two-stage reduction: $\mathcal{R}_1$ takes as input the tuple $(N, g, \mathcal{T})$ and computes $h \coloneqq g^{2^\mathcal{T}}$, then it sets $pp \coloneqq (\mathcal{T}, N, g, h)$ and runs $\mathcal{A}_1(pp)$, who outputs some message (τ, s_0, s_1). The output of $\mathcal{R}_1$ is the string (τ, s_0, s_1). The challenger provides $\mathcal{R}_2$ with the triple $(x, y, (\tau, s_0, s_1))$, who sets $u \coloneqq x$ and $v \coloneqq y \cdot s_b$ and runs $\mathcal{A}_2((u,v), \tau)$ and outputs whatever the adversary outputs. Observe that $\mathcal{R}_1$ is efficient, since $\mathcal{T}$ is a polynomial, and that the depth of $\mathcal{R}_2$ is close to that of $\mathcal{A}$. It is not hard to see that whenever $y = x^{2^\mathcal{T}}$ then reduction reproduces the distribution of $\mathcal{H}_1$, whereas if y is uniformly sampled in $\mathbb{J}_N$, then the simulation is identical to $\mathcal{H}_2$. Thus the success probability of $\mathcal{R}$ is identical to that of $\mathcal{A}$. This contradicts the sequential squaring assumption and bounds the difference between the two hybrids to a negligible factor.

Observe that in $\mathcal{H}_2$ the puzzle consists of two uniform elements of $\mathbb{J}_N$.

5.3 Public-Coin Setup

All of our schemes require a trusted setup where the random coins have to be kept private. If revealed, they would give one an unfair advantage in solving any puzzle. This does not seem to be an inherent limitation of the primitive and we could envision a dream-version of HTLPs where the setup can be run with public random coins. Towards this objective, one can generalize the techniques presented in Section 4.1 and Section 4.2 to hidden-order groups with public-coin setups [8], however this would hinder the efficiency of the schemes as

the tuple $(g, h = g^{2^{\mathcal{T}}})$ is no longer efficiently computable (by assumption). Depending on $\mathcal{T}$, this may require a significant initial investment in terms of computation.

Nevertheless, for certain applications (e.g., e-voting or sealed bid auctions) it might be perfectly acceptable to run $\mathcal{T}$ sequential squarings ahead of time to generate the tuple (g, h). Note that, in the variants described above, the puzzle is guaranteed to hide the payload for time proportional to $\mathcal{T}$, starting from the moment the puzzle is published. Therefore arbitrarily many puzzles can be efficiently spawned once (g, h) is fixed. Constructing an HTLP with an efficient public-coin setup is a fascinating open question.

5.4 Combining Puzzles of Different Hardness

Another limitation of our schemes is that the time parameter $\mathcal{T}$ is fixed once and for all in the setup. An easy solution to make our construction more flexible is to augment the setup with multiple $(\mathcal{T}_1, \ldots, \mathcal{T}_n)$. For the constructions in Section 4.1 and Section 4.2 is sufficient to set the public parameters as

$$pp \coloneqq \left(g, h_1 \coloneqq g^{2^{\mathcal{T}_1}}, \ldots, h_n \coloneqq g^{2^{\mathcal{T}_n}}\right)$$

which can be efficiently computed using the factors of N. Our scheme in Section 4.3 can also be extended by producing different obfuscated circuits $(MEvk^{(1)}, \ldots, MEvk^{(n)})$, with the appropriate $\mathcal{T}_i$ hardwired. Here it is important that the obfuscated circuits are sampled with fresh coins, so also the corresponding keys $(pk_0^{(1)}, \ldots, pk_0^{(n)})$ must be included in the setup.

It turns out that one can even combine puzzles generated with different parameters $\mathcal{T}_1$ and $\mathcal{T}_2$ in a natural way: Assume without loss of generality that $\mathcal{T}_1 > \mathcal{T}_2$, then clearly $2^{\mathcal{T}_2} \cdot \tilde{t} = 2^{\mathcal{T}_1}$, for some integer $\tilde{t} = 2^t$. Then the homomorphic evaluation over two puzzles (u_1, v_1) and (u_2, v_2) is done as follows

$$\tilde{u} \coloneqq u_1^{2^t} \cdot u_2 \pmod N \text{ and } \tilde{v} \coloneqq v_1 \cdot v_2 \pmod{N^2} \;/\; (\text{mod } N),$$

where the second modulus depends on whether we are considering linearly or multiplicatively homomorphic puzzles. Note that the hardness of the resulting puzzles $(\tilde{u}, \tilde{v})$ corresponds to the time proportional to solving it $(\mathcal{T}_2)$ + homomorphic evaluation $(t) = \mathcal{T}_1$. This is aligned with the expectation that the evaluation algorithm does not decrease the difficulty of a puzzle. For the fully-homomorphic construction the argument is a bit more delicate since the obfuscated circuits contain a trapdoor to efficiently solve the puzzles. Therefore, one has to ensure that the puzzles are re-encoded with the correct hardness parameter. This can be done via standard techniques, e.g., signing the puzzles and verifying the signatures inside the obfuscated circuits.

6 Applications

In this section we present some of the most interesting applications of HTLPs. We stress that our purpose is to demonstrate the usefulness of our primitive in broader contexts and not to construct systems that are ready to be deployed in practice. The precise implementation and the complete characterization of the security of such systems is beyond the scope of this work. In favor of a simpler presentation, we implicitly assume that all HTLPs are well-formed and all secrets are sampled from the correct domains. This can be always enforced by augmenting our schemes with non-interactive zero-knowledge proofs [13].

6.1 E-Voting

We construct an *e-voting* protocol with n voters and m candidates. An e-voting protocol consists of a *voting* phase and a *counting* phase and proceeds as follows: During the voting phase, each voter casts a vote for one of the candidates and the votes are counted during the subsequent counting phase. Finally the candidate with the largest amount of votes is announced as the winner of the election. The votes must be kept hidden for the duration of the first phase to avoid any bias.

Let $\mathcal{T}$ be the time bound of the *voting phase*. We propose an e-voting protocol based on our linearly homomorphic time-lock puzzle from Section 4.1. Here, the i-th vote, denoted by vote_i, consists of a tuple of m time-lock puzzles where the secret encoded is always 0 except at position j, where the secret is 1. This encodes the preference for the j-th candidate C_j. After receiving votes from all the voters, the puzzles are combined homomorphically to sum up the number of preferences for each candidate. We eventually obtain a final vote consisting of m puzzles, which are then solved to obtain the final vote tallies for each candidate.

Election Setup: Generate the public parameters $pp \leftarrow \mathsf{LHP.PSetup}(1^\lambda, \mathcal{T})$ and publish them so that they are accessible to all the voters.

Voting Phase: Each voter V_i, on deciding to vote the j-th candidate C_j (where $j \in \{1, \ldots, m\}$) does the following.

- For all $j' \in \{1, \ldots, m\}/j$, generate $Z_{j'} \leftarrow \mathsf{LHP.PGen}(pp, 0)$.
- Generate $Z_j \leftarrow \mathsf{LHP.PGen}(pp, 1)$.
- Compute $\mathsf{vote}_i = (Z_1, \ldots, Z_m)$ and output vote_i as the vote.

Counting Phase: Collect votes from all voters denoted by $(\mathsf{vote}_1, \ldots, \mathsf{vote}_n)$ and do the following.

- Parse each vote as $\mathsf{vote}_i = (Z_1^{(i)}, \ldots, Z_m^{(i)})$.
- For all $j \in \{1, \ldots, m\}$:
  - Compute the puzzle $\tilde{Z}_j \leftarrow \mathsf{LHP.PEval}(\oplus, pp, Z_j^{(1)}, \ldots, Z_j^{(n)})$.
  - Count the votes received by j-candidate by $v_j \leftarrow \mathsf{LHP.PSolve}(pp, \tilde{Z}_j)$.
- Output j^*-th candidate as the winner of the election, where $v_{j^*} = \mathsf{max}(v_1, \ldots, v_m)$.

By the security of LHTLP, the votes remain hidden for the whole duration of the voting phase. Furthermore, observe that we eventually need to only solve m puzzles, one puzzle per candidate. This is regardless on how many users go offline before the counting phase.

6.2 Multi-Party Coin Flipping

We consider the settings where n parties want to flip a coin in such a way that (i) the value of the coin is unbiased even if $n-1$ parties collude and (ii) all parties agree on the same value for the coin. Consider the protocol where parties commit to a bit and the result is the XOR of all the bits. The problem with this simple solution is that one party that controls the network traffic might learn all of the other bits and go offline if he does not agree with the outcome, thus biasing the result.

We propose the use of our linearly homomorphic time-lock puzzles to solve this problem. Let $\mathcal{T}$ be a bound on the runtime of the protocol. In our protocol, $\mathsf{LHP.PSetup}(1^\lambda, \mathcal{T})$ is run first to generate the public parameters pp. Then, every party P_i randomly chooses a bit $b_i \leftarrow_\$ \{0, 1\}$ and generates a time-lock puzzle as $Z_i \leftarrow \mathsf{LHP.PGen}(pp, b_i)$ before publishing it. Once P_i receives the puzzles from all other parties, it runs $Z \leftarrow \mathsf{LHP.PEval}(\oplus, pp, Z_1, \ldots, Z_n)$ to obtain the puzzle Z encoding the sum of all secrets. Each party P_i can solve Z to recover the corresponding s and output its least significant bit as the result of the coin flipping. Observe that only one puzzle needs to be solved regardless of the number of participants, even if everyone goes offline after the first phase. Since the time-lock puzzle is correct, then so is our protocol, furthermore the coins is unbiased by the security of LHTLP (in the timing model).

Setup: Generate the public parameters $pp \leftarrow \mathsf{LHP.PSetup}(1^\lambda, \mathcal{T})$ and publish them so that they are accessible to all the parties.

Coin Flipping: Each party P_i does the following.

- Choose $b_i \leftarrow_\$ \{0, 1\}$,
- Generate $Z_i \leftarrow \mathsf{LHP.PGen}(pp, b_i)$.

– Broadcast Z_i to all other parties.

Announcement of the Result: Each party P_i collects all the puzzles $Z_1, \ldots, Z_n$ from other parties and does the following.

– Compute the final puzzle $Z \leftarrow \mathsf{LHP.PEval}(\oplus, pp, Z_1, \ldots, Z_n)$.
– Solve the final puzzle as $s \leftarrow \mathsf{LHP.PSolve}(pp, Z)$
– Output $b \leftarrow \mathsf{LSB}(s)$ as the final result of the coin flipping.

6.3 Sealed Bid Auctions

Consider the settings where an auction is conducted with a set of n bidders $(B_1, \ldots, B_n)$. The bids are sealed throughout the bidding phase and disclosed during the opening phase. Once all of the bids are revealed, the highest bidder (or some other bidder depending on the allocation rule of the auction) is awarded as the winner. Sealed-bid auctions are one of the motivating examples for the usage of time-lock puzzles [6]. However, current solutions do not scale well with the amount of users going offline after the first phase.

To counter this issue we propose a protocol very similar to the coin-flipping one, where the setup generates the public parameters of the time-lock puzzles pp. In the bidding phase, each bidder generates a puzzle Z_i on input a bound $\mathcal{T}$ and his bid. The winner of the auction is the recovered by homomorphically evaluating the circuit Γ over all bids, where Γ computes the highest bid from a given list of bids and outputs the index of the corresponding bidder. Also in this case, only one puzzle has to be solved in the announcement phase. However, the function that needs to be homomorphically evaluated is no longer linear and therefore one needs to resort to fully-homomorphic time-lock puzzles (such as the scheme described in Section 4.3).

6.4 Multi-Party Contract Signing

Consider the settings where n mutually distrusting parties want to jointly sign a contract. The contract is enforceable only if signed by all parties. In a naive approach, a party P_i collects the signatures that were broadcast by all other parties and add its own to seal the contract. However, if P_i fails to broadcast its own signature, other parties are left empty-handed.

We propose a solution based on the combination of multiplicatively homomorphic time-lock puzzles (as described in Section 4.2) and RSA-aggregate signatures [20]. Loosely speaking, an aggregate signature scheme allows one to publicly combine signatures over different messages and under different keys in such a way that the digest is still efficiently verifiable. The crucial property of the construction of Hohenberger and Waters [20] is that signatures σ are elements of $\mathbb{QR}_N$, for some fixed RSA integer N, and the aggregation of $((pk_1, m_1, \sigma_1), \ldots, (pk_n, m_n, \sigma_n))$ is computed as

$$\sigma_{agg} = \prod_{j=1}^{n} \sigma_j \pmod N.$$

Since $\mathbb{QR}_N$ is a subgroup of $\mathbb{J}_N$, we can seamlessly compute the aggregation function homomorphically. Let M be the contract to be signed. Our contract-signing protocol proceeds as follows: In the setup phase, the public parameters of the Hohenberger-Waters signature scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign})$ and of MHTLP (with reusable setup) are generated. Note that we implicitly assume that both setup algorithms sample the same strong RSA integer N. Then we fix $\mathcal{T}_1 := \mathcal{T}$ for some fixed $\mathcal{T}$ (which is suggested to be in the order of $2^{30} - 2^{50}$ in [6]) and each $\mathcal{T}_i$ is defined as $\frac{\mathcal{T}_{i-1}}{2}$, until $\mathcal{T}_\ell := 2$. Each user generates a key pair (pk_i, sk_i) and enters in the following loop. In the k-th iteration, each party P_i generates a signature $\sigma_i^{(k)}$ on the contract M via the signing algorithm Sign[4]. Then it time-locks $\sigma_i^{(k)}$ with a timing hardness $\mathcal{T}_k$ via $Z_i^{(k)} \leftarrow \mathsf{MHP.PGen}(pp_2, \sigma_i^{(k)}, \mathcal{T}_k)$ and broadcasts $Z_i^{(k)}$. If every user successfully broadcasts $Z_i^{(k)}$, then the

[4] In [20] the signing algorithm requires an additional timing parameter, which we fix to be the round number and omit for the sake of clarity.

protocol proceeds to the next iteration. Otherwise each party collects the puzzles $(Z_1^{(k-1)}, \ldots, Z_n^{(k-1)})$ from the previous iteration and generates the final puzzle as $Z^{(k-1)} \leftarrow \mathsf{MHP.PEval}(\otimes, pp_2, Z_1^{(k-1)}, \ldots, Z_n^{(k-1)})$. Solving this final puzzle reveals the aggregated signature σ_{agg} on M.

Setup Phase: Generate the public parameters of the aggregate signature scheme as $pp_1 \leftarrow \mathsf{Setup}(1^\lambda, 1^T)$ and the public parameters of the time-lock puzzle MHTLP (with reusable setup and multiple hardness parameters) as $pp_2 \leftarrow \mathsf{MHP.PSetup}(1^\lambda, \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\ell)$ and broadcast it to all parties.

Key Generation Phase: Before the start of the first iteration, each party P_i executes the key generation algorithm $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(pp_1)$ to generate a public and private key pair (pk_i, sk_i).

Signing Phase: At the beginning of the k-th iteration, each party P_i does the following.

- Generate a signature on M as $\sigma_i^{(k)} \leftarrow \mathsf{Sign}(pp_1, sk_i, M)$.
- Time-lock the signature via $Z_i^{(k)} \leftarrow \mathsf{MHP.PGen}(pp_2, \sigma_i^{(k)}, \mathcal{T}_k)$ with timing hardness $\mathcal{T}_k$ and broadcast the puzzle.

Aggregation phase: If all parties had broadcast their puzzles, proceed to $(k+1)$-th iteration. If not (or if $k = \ell$), each party P_i does the following.

- Collect the puzzles $(Z_1^{(k-1)}, \ldots, Z_n^{(k-1)})$ from the $(k-1)$-th iteration.
- Generate the final puzzle as $Z^{(k-1)} \leftarrow \mathsf{MHP.PEval}(\otimes, pp_2, Z_1^{(k-1)}, \ldots, Z_n^{(k-1)})$.
- Solve the puzzle to obtain the aggregated signature $\sigma_{agg} \leftarrow \mathsf{MHP.PSolve}(pp_2, Z^{(k-1)})$ on M.
- Output (M, σ_{agg}).

Acknowledgements. Research supported in part by a gift from Ripple, a gift from DoS Networks, a grant from Northrop Grumman, a Cylab seed funding award, and a JP Morgan Faculty Fellowship.

References

1. Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015.
2. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.
3. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117. IEEE Computer Society Press, November 1982.
4. 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.
5. 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.
6. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Heidelberg, August 2000.
7. Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 1–20. Springer, Heidelberg, August 2010.
8. Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988.

9. Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.
10. Geoffroy Couteau, Thomas Peters, and David Pointcheval. Encryption switching protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 308–338. Springer, Heidelberg, August 2016.
11. Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.
12. Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st FOCS*, pages 283–293. IEEE Computer Society Press, November 2000.
13. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.
14. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.
15. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.
16. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.
17. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172. Springer, Heidelberg, August 2010.
18. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.
19. Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.
20. Susan Hohenberger and Brent Waters. Synchronized aggregate signatures from the RSA assumption. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 197–229. Springer, Heidelberg, April / May 2018.
21. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, Heidelberg, February 2007.
22. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.
23. Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In *58th FOCS*, pages 576–587. IEEE Computer Society Press, 2017.
24. Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 2018.
25. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50. Springer, Heidelberg, August 2011.
26. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388. ACM, January 2013.
27. Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2014.
28. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.
29. Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. https://eprint.iacr.org/2018/627.
30. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.
31. Benjamin Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. https://eprint.iacr.org/2018/623.

A Proof of Theorem 3

Proof. Our proof follows as a minor modification of the argument of Lemma 3.10 from [9]. For completeness, we show the analysis in the following. Let $\mathcal{A}$ be an adversary such that $\mathcal{A}$'s depth is bounded by $\mathcal{T}^{\varepsilon}(\lambda)$. We want to show that for every $\lambda \in \mathbb{N}$, the following holds:

$$\Pr\left[b \leftarrow \mathcal{A}_2(pp, Z) : \begin{array}{l} pp \leftarrow \mathsf{FHP.PSetup}(1^\lambda, \mathcal{T}(\lambda)) \\ b \leftarrow_{\$} \{0,1\} \\ Z \leftarrow \mathsf{FHP.PGen}(pp, b) \end{array}\right] \leq \frac{1}{2} + \nu(\lambda)$$

where $\nu(\cdot)$ is some negligible function. Towards this objective we define a sequence of hybrids.

Hybrid $\mathcal{H}^b_{\mathsf{real}}$: Defined as the original game. Note that the distribution induced by this hybrid, to they eyes of the adversary, is

$$\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{MProg}^{(sk_0,k,k')}), (\mathsf{PGen}(\mathcal{T}, b), \mathsf{Enc}(pk_0, b))\right).$$

Hybrid $\mathcal{H}^b_{\mathsf{inter}}$: Defined as the previous hybrid except that the time-lock puzzle of the challenge ciphertext is computed hardwiring the secret to 0. The corresponding distribution is

$$\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{MProg}^{(sk_0,k,k')}), \left(\boxed{\mathsf{PGen}(\mathcal{T}, 0)}, \mathsf{Enc}(pk_0, b)\right)\right).$$

It is easy to show that $\mathcal{H}^b_{\mathsf{real}}$ and $\mathcal{H}^b_{\mathsf{inter}}$ are computationally indistinguishable for an attacker of depth at most $\mathcal{T}^{\varepsilon}(\lambda)$ by an invocation of the security of the time-lock puzzle.

Hybrid $\mathcal{H}^b_{\mathsf{ideal}}$: Defined as the previous experiment except that the evaluation key is sampled by obfuscating the trapdoor circuit $\mathsf{tMProg}^{(k,k')}(i)$ defined as

$\underline{\mathsf{tMProg}^{(k,k')}(i):}$

$r_i \leftarrow \mathsf{PRF}(k, i), r'_i \leftarrow \mathsf{PRF}(k', i)$

$(tpk_i) \leftarrow \mathsf{tKeyGen}(1^\lambda; r_i)$

$tP_i \leftarrow \mathsf{tProg}^{(tpk_i)}$

$t\Lambda_i \leftarrow pi\mathcal{O}(1^p, tP_i; r'_i)$

return $(t\Lambda_i)$

and padded to the maximum circuit size. Thus, the view of the adversary is given by the following ensemble

$$\left(\mathcal{T}, pk_0, MEvk \leftarrow \boxed{i\mathcal{O}(1^p, \mathsf{tMProg}^{(k,k')})}, (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right).$$

Observe that in $\mathcal{H}^b_{\mathsf{ideal}}$ the view of the adversary is completely independent from the secret key sk_0 and therefore it holds that

$$\begin{aligned} &\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{tMProg}^{(k,k')}), (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, 0))\right) \\ \approx_\mu &\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{tMProg}^{(k,k')}), \left(\mathsf{PGen}(\mathcal{T}, 0), \boxed{\mathsf{Enc}(pk_0, 1)}\right)\right) \end{aligned}$$

by an invocation of the μ-hiding of the trapdoor encryption scheme. This implies that $\mathcal{H}^0_{\mathsf{ideal}}$ and $\mathcal{H}^1_{\mathsf{ideal}}$ are computationally indistinguishable to the eyes of the adversary. Thus, all it is left to be shown is that $\mathcal{H}^b_{\mathsf{inter}}$ and $\mathcal{H}^b_{\mathsf{ideal}}$ are computationally indistinguishable. Towards this goal, consider the following sequence of intermediate hybrids.

Hybrid $\mathcal{H}^b_\ell$ for $0 \leq \ell \leq \mathcal{L}$: These hybrids are identical to $\mathcal{H}^b_{\mathsf{inter}}$ except that $MEvk$ is generated in a different way. More specifically, instead of obfuscating the honest master evaluation program $\mathsf{MProg}^{(sk_0,k,k')}$, obfuscate $\mathsf{hMProg}^{(sk_0,k,k')}_\ell(i)$ (padded to the maximum circuit size) defined as

$\mathsf{hMProg}_{\ell}^{(sk_0,k,k')}(i)$:

if $i < \mathcal{L} - \ell$ **then**
  $\Lambda_i \leftarrow \mathsf{MProg}^{(sk_0,k,k')}$
  return Λ_i
elseif $i > \mathcal{L} - \ell$ **then**
  $t\Lambda_i \leftarrow \mathsf{tMProg}^{(k,k')}$
  return $t\Lambda_i$
elseif $i = \mathcal{L} - \ell$ **then**
  $r_{i-1} \leftarrow \mathsf{PRF}(k, i-1), r_i \leftarrow \mathsf{PRF}(k, i), r'_i \leftarrow \mathsf{PRF}(k', i)$
  $(pk_{i-1}, sk_{i-1}) \leftarrow \mathsf{KeyGen}(1^\lambda; r_{i-1})$
  $\boxed{tpk_i \leftarrow \mathsf{tKeyGen}(1^\lambda, r_i)}$
  $P_i \leftarrow \mathsf{Prog}^{(sk_{i-1}, tpk_i)}$
  $h\Lambda_i \leftarrow pi\mathcal{O}(1^p, P_i; r'_i)$
  return $(h\Lambda_i)$
endif

The distribution induced by each hybrid is

$$\left(\mathcal{T}, pk_0, MEvk_\ell \leftarrow \boxed{i\mathcal{O}(1^p, \mathsf{hMProg}_{\ell}^{(sk_0,k,k')})}, (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right)$$

where pk_0, sk_0, k, k' are all randomly sampled. Observe that $\mathsf{hMProg}_0^{(sk_0,k,k')})$ is functionally equivalent to $\mathsf{MProg}^{(sk_0,k,k')})$ and therefore

$$\begin{aligned}&\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{MProg}^{(sk_0,k,k')}), (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right)\\ \approx_\mu &\left(\mathcal{T}, pk_0, MEvk \leftarrow \boxed{i\mathcal{O}(1^p, \mathsf{hMProg}_0^{(sk_0,k,k')})}, (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right)\end{aligned}$$

by an invocation of the μ-indistinguishability of $i\mathcal{O}$. It follows that $\mathcal{H}^b_{\mathsf{inter}}$ and $\mathcal{H}^b_0$ are indistinguishable. The same argument applies to $\mathcal{H}^b_\ell$ and $\mathcal{H}^b_{\mathsf{ideal}}$. Therefore it suffices to show that the neighboring hybrids are computationally close to the eyes of the adversary. This is captured by the following claim.

Claim. For all $b \in \{0,1\}$ and all $0 \le \ell < \mathcal{L}$, $\mathcal{H}^b_i$ and $\mathcal{H}^b_{i+1}$ are μ-indistinguishable.

Fix some $b \in \{0,1\}$ and some $0 \le \ell < \mathcal{L}$. To prove the above claim one needs to show that

$$\begin{aligned}&\left(\mathcal{T}, pk_0, MEvk \leftarrow i\mathcal{O}(1^p, \mathsf{hMProg}_{\ell}^{(sk_0,k,k')}), (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right)\\ \approx_\mu &\left(\mathcal{T}, pk_0, MEvk \leftarrow \boxed{i\mathcal{O}(1^p, \mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')})}, (\mathsf{PGen}(\mathcal{T}, 0), \mathsf{Enc}(pk_0, b))\right).\end{aligned}$$

The only difference between the views of $\mathcal{A}$ lies in which hybrid master program is obfuscated, furthermore, notice that $\mathsf{hMProg}_{\ell}^{(sk_0,k,k')}$ and $\mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')}$ output the same at all points except at two inputs $x = \mathcal{L} - \ell$ and $x - 1 = \mathcal{L} - \ell - 1$:

- At x, $\mathsf{hMProg}_{\ell}^{(sk_0,k,k')}$ outputs a $pi\mathcal{O}$ obfuscation of the hybrid program $\mathsf{Prog}^{(sk_{x-1}, tpk_x)}$, whereas the program $\mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')}$ outputs an obfuscation of the trapdoor program $\mathsf{tProg}^{(tpk_x)}$, all random variables are generated with randomness output by the PRF's using k, k'.

– At $x-1$, $\mathsf{hMProg}_{\ell}^{(sk_0,k,k')}$ outputs an obfuscation of the honest program $\mathsf{Prog}^{(sk_{x-2},pk_{x-1})}$, whereas $\mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')}$ outputs an obfuscation of the hybrid program $\mathsf{Prog}^{(sk_{x-2},tpk_{x-1})}$.

With this in mind, we consider another sequence of hybrids $\mathcal{G}_0,\ldots,\mathcal{G}_6$ where $\mathcal{G}_0$ samples obfuscations of programs $\Gamma_0 = \mathsf{hMProg}_{\ell}^{(sk_0,k,k')}$ and $\mathcal{G}_6$ samples obfuscations of programs $\Gamma_6 = \mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')}$. The intermediate hybrids namely $\mathcal{G}_i$ produces obfuscations of a hybrid program Γ_i in the following way:

Hybrid $\mathcal{G}_1$: In this hybrid a program Γ_1 is constructed as follows: After sampling two PRF keys k and k', puncture the PRF key k at points x and $x-1$ and k' at x:

$$k(x,x-1) \leftarrow \mathsf{Puncture}(k,\{x,x-1\}), k'(x) \leftarrow \mathsf{Puncture}(k',x)$$

and let r_{x-1}, r_x and r'_x be the outputs of the PRF computed as:

$$r_{x-1} \leftarrow \mathsf{PRF}(k,x-1), r_x \leftarrow \mathsf{PRF}(k,x), r'_x \leftarrow \mathsf{PRF}(k',x)$$

Note that in Γ_0, r_{x-1} and r_x are used only to generate keys for layers $x-1$ and x, and r'_x is used to generate the x-th layer evaluation key. Directly computing the variables that depend on r_{x-1}, r_x, r'_x:

$$\begin{aligned}(pk_{x-1},sk_{x-1}) &\leftarrow \mathsf{KeyGen}(1^\lambda; r_{x-1}),\\ tpk_x &\leftarrow \mathsf{tKeyGen}(1^\lambda, r_x),\\ h\Lambda_x &\leftarrow pi\mathcal{O}(1^p, P_x; r'_x)\end{aligned}$$

where $P_x = \mathsf{Prog}^{(sk_{x-1},tpk_x)}$. The program Γ_1 is identical to Γ_0, except for: (1) instead of having k,k' hardwired, Γ_1 has the punctured keys $k(x,x-1)$ and $k'(x)$ hardwired in, along with $pk_{x-1}, h\Lambda_x$ (2) Γ_1 proceeds the same way as Γ_0 for all inputs except for $x-1$ and x; for input x, it directly outputs $h\Lambda_x$, and for $x-1$, it uses the hardwired key pk_{x-1} to compute Λ_{x-1} as in Γ_0. Since Γ_0 and Γ_1 have the same functionality, it follows from the μ-indistinguishability of $i\mathcal{O}$ that their obfuscation is μ-indistinguishable.

Hybrid $\mathcal{G}_2$: Hybrid $\mathcal{G}_2$ proceeds identically to $\mathcal{G}_1$, except that it computes the keys and obfuscated program to be hardwired using true randomness instead of pseudorandom coins, that is,

$$\begin{aligned}(pk_{x-1},sk_{x-1}) &\leftarrow \mathsf{KeyGen}(1^\lambda),\\ tpk_x &\leftarrow \mathsf{tKeyGen}(1^\lambda),\\ h\Lambda_x &\leftarrow pi\mathcal{O}(1^p, P_x)\end{aligned}$$

where $P_x = \mathsf{Prog}^{(sk_{x-1},tpk_x)}$. $\mathcal{G}_2$ then obfuscates the program Γ_2 that is identical to Γ_1 except that it contains pk_{x-1} and $h\Lambda_x$ generated using true randomness as above. Since the puncturable PRF is pseudorandom, it follows that $\mathcal{G}_2$ and $\mathcal{G}_1$ are μ-indistinguishable.

Hybrid $\mathcal{G}_3$: Hybrid $\mathcal{G}_3$ is identical to $\mathcal{G}_2$ except it samples $h\Lambda_x$ that is an obfuscation of the trapdoor program $tP_x \coloneqq \mathsf{tProg}^{(tpk_x)}$. That is,

$$t\Lambda_x \leftarrow pi\mathcal{O}(1^p, tP_x)$$

$\mathcal{G}_3$ then obfuscates Γ_3 that is identical to Γ_2 except that $t\Lambda_x$ (together with pk_{x-1}) is hardwired instead of $h\Lambda_x$ in $\mathcal{G}_2$. Indistinguishability follows from the security of $pi\mathcal{O}$ for the class of $\mathbf{S}$. More precisely, consider any fixed sequence of $\{pk_{x-1}, sk_{x-1}\}$ and let $SK = \{sk_{x-1}\}$. The distribution D^{SK} samples the following tuple:

$$(P_x = \mathsf{Prog}^{(sk_{x-1},pk_x)}, tP_x = \mathsf{tProg}^{(tpk_x)}, z = tpk_x) \leftarrow D_p^{SK}$$

Thus by the security of $pi\mathcal{O}$ with respect to D^{SK}, the following is the case:

$$(P_x, tP_x, h\Lambda_x \leftarrow pi\mathcal{O}(1^p, P_x), tpk_x) \approx_\mu (P_x, tP_x, t\Lambda_x \leftarrow pi\mathcal{O}(1^p, tP_x), tpk_x)$$

Since this holds for all sequence of $\{pk_{x-1}, sk_{x-1}\}$, it directly implies that $\mathcal{G}_2$ and $\mathcal{G}_3$ are μ-indistinguishable.

Hybrid $\mathcal{G}_4$: The hybrid obfuscates the program Γ_4 that works the same way as Γ_3 except that the hardwired honest public key pk_{x-1} is replaced by the trapdoor public key $tpk_{x-1} \leftarrow \mathsf{tKeyGen}(1^\lambda)$. The indistinguishability of the hybrids follows from the μ-indistinguishability of the public and the trapdoor keys of Π.

Hybrid $\mathcal{G}_5$: The hybrid obfuscates the program Γ_5 that is identical to Γ_4 except that the values $tpk_{x-1}, t\Lambda_x$ hardwired in Γ_5 are generated using pseudorandom strings r_{x-1}, r_x, r'_x:

$$\begin{aligned} tpk_{x-1} &\leftarrow \mathsf{tKeyGen}(1^\lambda; r_{x-1}), \\ tpk_x &\leftarrow \mathsf{tKeyGen}(1^\lambda, r_x), \\ t\Lambda_x &\leftarrow pi\mathcal{O}(1^p, tP_x; r'_x) \end{aligned}$$

where $tP_x = \mathsf{tProg}^{(tpk_x)}$. Since the puncturable PRF is pseudorandom, it follows that $\mathcal{G}_5$ and $\mathcal{G}_4$ are μ-indistinguishable.

Hybrid $\mathcal{G}_6$: The hybrid outputs an obfuscation of $\Gamma_6 = \mathsf{hMProg}_{\ell+1}^{(sk_0,k,k')}$. Since Γ_6 has the same functionality as Γ_5, from the indistinguishability security of $i\mathcal{O}$, it follows that $\mathcal{G}_5$ is μ-indistinguishable from $\mathcal{G}_6$. Therefore, by hybrid argument, the experiment $\mathcal{H}_\ell^b$ and $\mathcal{H}_{\ell+1}^b$ are μ'-indistinguishable, where $\mu'(\lambda) = c \cdot \mu(\lambda)$ for some constant c. This proves the claim and concludes our proof. □