# Generic Constructions of Incremental and Homomorphic Timed-Release Encryption

Peter Chvojka[1], Tibor Jager[1],
Daniel Slamanig[2], and Christoph Striecks[2]

[1] University of Wuppertal, Germany
{chvojka,tibor.jager}@uni-wuppertal.de
[2] AIT Austrian Institute of Technology, Vienna, Austria
firstname.lastname@ait.ac.at

**Abstract.** Timed-release encryption (TRE) makes it possible to send information "into the future" such that a pre-determined amount of time needs to pass before the information can be decrypted, which has found numerous applications. The most prominent construction is based on sequential squaring in RSA groups, proposed by Rivest *et al.* in 1996. Malavolta and Thyagarajan (CRYPTO'19) recently proposed an interesting variant of TRE called homomorphic time-lock puzzles (HTLPs). Here one considers multiple puzzles which can be independently generated by different entities. One can homomorphically evaluate a circuit over these puzzles to obtain a new puzzle. Solving this new puzzle yields the output of a circuit evaluated on all solutions of the original puzzles. While this is an interesting concept and enables various new applications, for constructions under standard assumptions one has to rely on sequential squaring.

We observe that viewing HTLPs as homomorphic TRE gives rise to a simple generic construction that avoids the homomorphic evaluation on the puzzles and thus the restriction of relying on sequential squaring. It can be instantiated based on any TLP, such as those based on one-way functions and the LWE assumption (via randomized encodings), while providing essentially the same functionality for applications. Moreover, it overcomes the limitation of the approach of Malavolta and Thyagarajan that, despite the homomorphism, one puzzle needs to be solved per decrypted ciphertext. Hence, we obtain a *"solve one, get many for free"* property for an arbitrary amount of encrypted data, as we only need to solve a single puzzle independent of the number of ciphertexts. In addition, we introduce the notion of incremental TLPs as a particularly useful generalization of TLPs, which yields particularly practical (homomorphic) TRE schemes. Finally, we demonstrate various applications by firstly showcasing their cryptographic application to construct dual variants of timed-release functional encryption and also show that we can instantiate previous applications of HTLPs in a simpler and more efficient way.

## 1 Introduction

Timed-release encryption (TRE) has the goal of sending information into the future in a way that the sender can be sure that a pre-determined amount of time needs to pass before the information can be decrypted. This idea was firstly discussed by May [May93], who introduced this notion and proposed a solution based on trusted agents. The idea is to rely on some trusted entity, which after the pre-determined time has passed, releases some secret that allows to efficiently obtain the hidden information (e.g., [DOR99,CHKO08]). In this paper we will not focus on this agent-based variant of TRE, but rather on an alternative idea proposed by Rivest *et al.* in [RSW96] and which relies on so called time-lock puzzles (TLPs). TLPs allow to seal messages in such a way that one is able to obtain the sealed message only by executing an expensive *sequential* computation. Thereby, the amount of time required to perform this sequential computation is a hardness parameter of the TLP which can be freely chosen. This approach does not involve a trusted agent and a sender can just publish a puzzle whose solution is the hidden message, and this message stays hidden until enough time has elapsed for the puzzle to be solved. TLPs have found numerous applications such as sealed-bid auctions [RSW96], fair contract signing [BN00], zero-knowledge arguments [DN00], or non-malleable commitments [LPS17].

The solution proposed by Rivest *et al.* in [RSW96] uses iterated squaring in an RSA group $\mathbb{Z}_N^*$ where $N$ it the product of two large primes $p$ and $q$. More precisely, a time lock puzzle $Z$ with solution $s$ for hardness $T$ is defined as $Z = (N, T, x, x^{2^T} \cdot k, \mathsf{Enc}(k, s))$ where $(x, k) \xleftarrow{\$} (\mathbb{Z}_N^*)^2$ and $\mathsf{Enc}$ is a symmetric encryption scheme. Note that this TLP can be equivalently viewed as a timed-release encryption (TRE) that encrypts message $s$. An interesting feature of this TLP construction is that creating a puzzle is much faster than the expensive sequential computation to solve the puzzle, i.e., knowing the factorization of $N$ and thus $\varphi(N)$ one can compute $x^{2^T}$ by first computing $2^T \bmod \varphi(N)$. This is an important property of TLPs when the required amount of time until the puzzle should be solved is very large. Interestingly, TLPs with this property seem hard to find. In [MMV11] Mahmoody *et al.* show that in the random-oracle model it is impossible to construct TLPs from one-way permutations and collision-resistant hash-functions that require more parallel time to solve than the total work required to generate a puzzle and thus ruling out black-box constructions of such TLPs. On the positive side, Bitansky *et al.* [BGJ+16] show how to construct TLPs with the aforementioned property from randomized encodings [IK00,AIK06]. However, this approach relies on indistinguishability obfuscation. Interestingly, when slightly relaxing the requirements and allowing efficient parallel computation in the generation (so-called *weak* TLPs) of the puzzles or a solution independent preprocessing, then such TLPs can be constructed generically from one-way functions and the learning with errors (LWE) assumption, respectively, via randomized encodings.

Recently, Malavolta and Thyagarajan [MT19] proposed an interesting variant of TLPs called *homomorphic* TLPs (HTLPs). Here one considers multiple puzzles $(Z_1, \ldots, Z_n)$ which can be independently generated by different entities

and without knowing the corresponding solutions $(s_1, \ldots, s_n)$ one can homomorphically evaluate a circuit $C$ over these puzzles to obtain as result a puzzle $\widehat{Z}$ with solution $C(s_1, \ldots, s_n)$, where the hardness of this resulting puzzle does not depend on the size of the circuit $C$ that was evaluated (which is called compactness). Consequently, this allows to aggregate a potentially large number of puzzles in a way that only a single puzzle needs to be solved. While this concept is interesting on its own, Malavolta and Thyagarajan [MT19] also show that it extends the applications of TLPs and in particular present applications to e-voting, multi-party coin flipping as well as multi-party contract signing. Moreover, it is reasonable to conjecture (as done by Malavolta and Thyagarajan) that any application that involves a large number of users and thus the constraint of requiring to solve multiple puzzles constitute one of the main obstacles that so far prevented the large scale adoption of TLPs. And exactly this problem can be overcome with HTLPs. Later Brakerski *et al.* in [BDGM19] further studied the concept of HTLPs and proposed the first fully homomorphic TLP (FHTLP) from standard assumption (whereas the FHTLP in [MT19] required the existence of sub-exponentially hard indistinguishability obfuscation). In contrast to the basic definition of HTLPs where the time required to solve the puzzles starts with the generation of the parameters, latter construction achieves an alternative notion where the time starts to run for every single puzzle at the point where the puzzle is generated.

### 1.1 Motivation for our Work

Our motivation stems from some observations about the approach to HTLPs by Malavolta and Thyagarajan [MT19]. Loosely speaking, they construct a linearly homomorphic TLP (LHTLP) from the iterated squaring TLP and Paillier encryption [Pai99]. They set up public parameters $(N, T, g, h = g^{2^T})$ for a suitable choice of $g$ and to create a puzzle to solution $s$, one re-randomizes $g, h$ for fresh $r \xleftarrow{\$} [N^2]$ and sets $Z = (g^r \pmod N, h^{r \cdot N}(1+N)^s \pmod{N^2})$. It is easy to see that this puzzle is linearly homomorphic where the evaluation is independent of the hardness $T$ (and one can also turn this into a multiplicatively homomorphic TLP). Now this is the basis for all applications (apart from the FHTLP which requires indistinguishability obfuscation) and also the FHTLP due to Brakerski *et al.* in [BDGM19] requires an LHTLP (where to the best of our knowledge the aforementioned is the only known construction). So our first observation is that we can equivalently view their HTLPs as homomorphic timed-release encryption (HTRE), i.e., a TRE scheme that supports homomorphic evaluation on the encrypted messages. Secondly, all these constructions are not generic as they rely on a single particular construction of an HTLP from sequential squaring. Moreover, one can see that for every such puzzle one can only start to attempt to solve it when $g^r$ is available as solving it requires sequentially computing $(g^r)^{2^T}$. The same also holds for the puzzle obtained from homomorphically evaluating on many such puzzles. Thirdly, while the homomorphic property makes it scalable in a setting where one is only interested in the homomorphic evaluation over

all encrypted messages, it would be convenient to have an approach that also supports this *"solve one, get many for free"* property even if one wants to obtain all encrypted messages instead of only the result of the homomorphic evaluation. Our final observation is that for all the applications discussed in [MT19] it seems sufficient, and in some applications even more desirable, when the runtime of the puzzle is counted from the point of running the puzzle setup algorithm. For instance, in the e-voting application from [MT19] it rather seems to complicate issues when the puzzle only starts after the last voter cast its vote. And even if this is not required, it might be easy to adjust the setup in a way that it outputs a set of public parameters, and a user can choose which public parameters to use when computing a puzzle.

Motivated by these observations, we ask whether it is possible to come up with an alternative approach that allows to have multiple puzzles computed by different parties, evaluate functions on their solutions and also only require solving a single puzzle, i.e., provide this *"solve one, get many for free"* property for homomorphic evaluations on many messages but also when it is required to decrypt all single messages. Ideally this approach is generic in nature and thus would allow to construct (homomorphic) timed-release encryption (TRE) generically from *any* TLP.

### 1.2 Our Contributions and Techniques

In this paper, among other contributions, we answer the aforementioned question to the affirmative. Subsequently, we summarize our contributions and the techniques used to obtain them.

*Generic (Homomorphic) Timed-Release Encryption.* We recall that we can equivalently view the construction of HTLPs by Malavolta and Thyagarajan in [MT19] as homomorphic timed-release encryption (HTRE). This observation allows us to come up with an approach that is orthogonal to the one described by Malavolta and Thyagarajan and enables us to encrypt an arbitrary number of messages with respect to one puzzle. Nevertheless, we thereby achieve the same goals. The basic and indeed very simple idea is that given *any* TLP we can use it to generate a puzzle $Z$ and its solution $s$, and we can use $s$ as the random coins for the key generation algorithm $\mathsf{Gen}(1^\lambda; s)$ of a public key encryption scheme. Then, we provide the respective public key $\mathsf{pk}$ as parameters of the TRE and solving the puzzle $Z$ reveals $s$ and thus $\mathsf{sk}$ allowing to decrypt *all* of the ciphertexts computed with respect to $\mathsf{pk}$. We note that in contrast to Malavolta and Thyagarajan who need to rely on their specific HTLP from sequential squaring (when relying on standard assumptions and avoiding indistinguishability obfuscation), our approach can also be instantiated from TLPs based on one-way functions and the LWE assumption (via randomized encodings). Using $s$ as the random coins for a (fully) homomorphic encryption scheme, immediately yields (fully) homomorphic TRE. Interestingly, this approach then allows us to obtain the *"solve one, get many for free"* property for both, the result of a homomorphic evaluation of many ciphertexts, but also if we want to decrypt all ciphertexts

individually. While this property for homomorphic evaluation provided by Malavolta and Thyagarajan is advantageous over solving many puzzles especially if one is interested in some aggregated data over all the ciphertexts, we believe that our solution offers additional benefits. In particular, it allows when solving one puzzle to decrypt all ciphertexts from this time period, independent of the encrypted amount of data. Thus, it would also support complex functions after decryption for which constructions of HTLPs are practically inefficient. Note that in the case of TLPs from sequential squarings, we can then also combine this with a single CPU (e.g., running on a public server) that computes and outputs the result of the sequential squarings, such that the amortized complexity is minimal, which is particularly interesting for incremental TLPs discussed below.

*Incremental (Homomorphic) Timed-Release Encryption.* We introduce the notion of incremental TLPs as a particularly useful generalization of TLPs, which yields particularly practical TRE schemes. In contrast to our basic TRE, however, instantiations need to rely on TLPs from the sequential squaring assumption. The basic idea is that the puzzle generation takes a sequence of hardness parameters $T_1, \ldots, T_n$ (where we assume that $T_i < T_{i+1}$ for all $i \in [n-1]$) and outputs a sequence of puzzles and solutions $(Z_i, s_i)_{i \in [n]}$. Now the distinguishing features is that puzzles can be solved incrementally in a way that solving $Z_i$ additionally considers solution $s_{i-1}$ and the time required to solve puzzle $Z_i$ is determined by the hardness $T_i - T_{i-1}$ (note that having $n = 1$ this yields a conventional TLP). This is interesting since one can use a *single* centralized server that continuously computes and publishes solutions to decrypt an arbitrary number of ciphertexts. Most importantly, the server would be *independent* of these ciphertexts, which is not achieved by prior constructions. Moreover, the decrypting parties would not have to solve *any puzzle at all*, but merely would have to wait until the server publishes a solution.

Since incremental TLPs are just a variant of TLPs, we can use our generic TRE framework to construct incremental (homomorphic) TRE schemes following the same ideas as outlined above, where homomorphic computations based on a conventional homomorphic encryption scheme are only supported within a time period. If one requires computations among different time periods, this can be achieved by using a multi-key (fully) homomorphic encryption scheme [LTV12] instead.

*Applications.* We present two types of applications of our TRE framework. Firstly, we demonstrate that our TRE framework can be used to obtain other more powerful variants of TRE in a generic way. Therefore, we showcase this using the regime of functional encryption. Recall, that in a functional encryption scheme decryption keys are associated to functions $f$ and ciphertexts are computed with respect to some public key $\mathsf{pk}$. Given a ciphertext $c = \mathsf{Enc}(\mathsf{pk}, x)$, a secret key $\mathsf{sk}_f$ for function $f$ allows to compute a decryption of $f(x)$ but reveals nothing beyond this about the encrypted message $x$. We now propose two variants which we call functional timed-release encryption (FTRE) as well

as timed-release functional encryption (TRFE). Loosely speaking, in FTRE we time-lock a function $f$ and after a certain time has passed everyone can learn the function $f$ of any ever encrypted message $x$. In TRFE on the other hand, messages $x$ are locked in a way that after a certain time has passed anyone in possession of a secret key for any function $f$ can learn $f(x)$. We discuss two applications based on incremental TRE. Firstly, an FTRE instantiation of identity-based encryption (IBE) [BF01] with locked keys, where the key generator at registration gives locked IBE secret keys for various validity periods (e.g., each for a month) to the user and the respective secret keys then unlock over time. Secondly, an TRFE instantiation of timed-release inner-product functional encryption (IPFE) [ABDP15], where it can be guaranteed that statistical analysis or encrypted data is only feasible after a certain amount of time has passed.

Secondly, we investigate existing applications of HTLPs and in particular the applications proposed by Malavolta and Thyagarajan. Concretely, using our approach all their applications can be redesigned using our TRE which leads to more efficient protocols without requirement to use homomorphic evaluation on puzzles.

**Notation.** We denote our security parameter as $\lambda$. For all $n \in \mathbb{N}$, we denote by $1^n$ the $n$-bit string of all ones. For any element $x$ in a set $S$, we use $x \xleftarrow{\$} S$ to indicate that we choose $x$ uniformly random from $S$. All algorithms may be randomized. For any PPT algorithm $A$, we define $x \leftarrow A(1^\lambda, a_1, \ldots, a_n)$ as the execution of $A$ with inputs security parameter $\lambda, a_1, \ldots, a_n$ and fresh randomness and then assigning the output to $x$ (we will usually omit $\lambda$ and assume that all algorithms take $\lambda$ as input). We use the notation $[n]$ to denote the set $\{1, \ldots, n\}$. For set $\{a_1, \ldots, a_n\}$ we use notation $(a_i)_{i\in[n]}$ and in similar way we use this notation also for set of tuples. We write $(x_i \leftarrow A(\mathsf{input}_i))_{i\in[n]}$ to denote running $n$ times the algorithm $A$ with fresh randomness on inputs $\mathsf{input}_1, \ldots, \mathsf{input}_n$ and assigning the output to $x_1, \ldots, x_n$. We use $\mathsf{poly}(\cdot)$ to denote some polynomial and $\mathsf{polylog}(\cdot)$ to denote a polylogarithmic function.

**Outline.** In Section 2 we present our definition for time-lock puzzles, variants of existing time-lock puzzles and discuss how they are related to our notion. We then introduce incremental time-lock puzzles and provide instantiations of (incremental) time-lock puzzles from sequential squarings and randomized encodings. Section 3 presents our generic construction of timed-release encryption, also covering incremental and homomorphic variants thereof. Finally, in Section 4 we discuss cryptographic as well as real-world applications of our generic TRE framework.

## 2 Time-Lock Puzzles

The terms time-lock puzzle [RSW96], timed-release encryption [May93], and time-lock encryption [LJKW18] are often interchangeably used in the literature

to denote an encryption scheme which enables us to send messages in the future. In this work we use the former two notions in a slightly different way, and hence we need to distinguish between them.

**Timed-release encryption** will denote an encryption scheme which allows us send messages "into the future", while providing confidentiality of message before the release time. This scheme may have additional properties, such as being additively/multiplicatively/fully homomorphic.

**Time-lock puzzles** provide the core functionality of a puzzle that needs a certain amount of time to be solved, and will be used as a building block for timed-release encryption.

### 2.1 Simple Time-Lock Puzzles

In this section we give a new definition for time-lock puzzles and explain how it relates to the old definition.

**Definition 1.** *A time-lock puzzle is pair of algorithms* $\mathtt{TLP} = (\mathsf{Gen}, \mathsf{Solve})$ *with the following syntax.*

- $(Z, s) \leftarrow \mathsf{Gen}(T)$ *is a probabilistic algorithm which takes as input a hardness parameter* $T \in \mathbb{N}$ *and outputs a puzzle* $Z$ *together with the unique solution* $s$ *of the puzzle. We require that* $\mathsf{Gen}$ *runs in time at most* $\mathsf{poly}(\log T, \lambda)$ *for some polynomial* $\mathsf{poly}$.
- $s \leftarrow \mathsf{Solve}(Z)$ *is a deterministic algorithm which takes as input a puzzle* $Z$ *and outputs a solution* $s \in S$*, where* $S$ *is a finite set. We require that* $\mathsf{Solve}$ *runs in time at most* $T \cdot \mathsf{poly}(\lambda)$*. There will also be a lower bound on the running time, which is part of the security definition.*

*We say* $\mathtt{TLP}$ *is* correct *if for all* $\lambda \in \mathbb{N}$ *and for all polynomials* $T$ *in* $\lambda$ *it holds:*

$$\Pr[s = s' : (Z, s) \leftarrow \mathsf{Gen}(T), s' \leftarrow \mathsf{Solve}(Z)] = 1.$$

*Relation to prior definitions.* In the definitions of time-lock puzzles from Bitansky *et al.* [BGJ+16] and Malavolta and Thyagarajan [MT19] algorithm $\mathsf{Gen}$ receives $s$ as an additional input and output a puzzle $Z$. This immediately yields a timed-release encryption scheme by viewing $s$ as a message that is encrypted. Our definition enables a slightly simpler generic construction of (homomorphic) timed-release encryption. Intuitively, our new definitions relates to the prior one in a similar way like a key encapsulation mechanism relates to an encryption scheme. Concretely, let $\mathtt{TLP} = (\mathsf{Gen}, \mathsf{Solve})$ be a puzzle according to our new definition. Then we obtain a puzzle $\mathtt{TLP}' = (\mathsf{Gen}', \mathsf{Solve}')$ of the old form as follows:

- $\mathsf{Gen}'(T, m)$ computes $Z \leftarrow \mathsf{Gen}(T)$ outputs $Z' = (Z, m \oplus s)$.
- $\mathsf{Solve}'(Z' = (Z, c))$ computes $s \leftarrow \mathsf{Solve}(Z)$ and outputs $c \oplus s$.

*Security.* For security we require that the solution of a time lock puzzle is indistinguishable from random, unless the adversary has sufficient running time to solve the puzzle. The following definition is inspired by those from Bitansky *et al.* [BGJ+16] and Malavolta and Thyagarajan [MT19], but adopted to our slightly modified definition of the $\mathsf{Gen}$ algorithm.

**Definition 2.** *Consider the security experiment* $\mathsf{ExpTLP}^b_{\mathcal{A}}(1^\lambda)$ *in Figure 1. We say that a time lock puzzle* $\mathtt{TLP}$ *is* secure with gap $\epsilon < 1$*, if there exists a polynomial* $\tilde{T}(\cdot)$ *such that for all polynomials* $T(\cdot) \geq \tilde{T}(\cdot)$ *and every polynomial-size adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ *of depth* $\leq T^\epsilon(\lambda)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *it holds*

$$\mathbf{Adv}^{\mathsf{TLP}}_{\mathcal{A}} = \left|\Pr\left[\mathsf{ExpTLP}^0_{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\mathsf{ExpTLP}^1_{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda).$$

**Other Variants of TLPs.** Here we briefly discuss weaker forms of TLPs as introduced by Bitansky *et al.* [BGJ+16]. First, weak TLPs (wTLPs) that do not require that $\mathsf{Gen}$ can be computed in time $\mathsf{poly}(\log T, \lambda)$, but either in fast parallel time ($\mathsf{Gen}$ can be computed by a uniform circuit of size $\mathsf{poly}(T, \lambda)$ and depth $\mathsf{poly}(\log T, \lambda)$) or there can be an (expensive) setup independent of the solution $s$ and $\mathsf{Gen}$ then runs in (sequential) time $\mathsf{poly}(\log T, \lambda)$.

**Definition 3 (Weak Time-Lock Puzzles [BGJ+16]).** *A weak time-lock puzzle (wTLP)* $\mathtt{wTLP} = (\mathsf{Gen}, \mathsf{Solve})$ *is satisfying the syntax and completeness requirements as per Definition 1, but with the following efficiency requirements:* $\mathsf{Gen}$ *can be computed by a uniform circuit of size* $\mathsf{poly}(T, \lambda)$ *and depth* $\mathsf{poly}(\log T, \lambda)$ *and* $\mathsf{Solve}$ *can be computed in time* $T \cdot \mathsf{poly}(\lambda)$*.*

**Definition 4 (Time-Lock Puzzles with Pre-processing [BGJ+16]).** *A time-lock puzzle with pre-processing (ppTLP) is a tuple of algorithms* $\mathtt{ppTLP} = (\mathsf{Preproc}, \mathsf{Gen}, \mathsf{Solve})$*:*

- $(P, K) \leftarrow \mathsf{Preproc}(T)$ *is a probabilistic algorithm that takes as input a difficulty parameter* $T$ *and outputs a state* $P$ *and a short* $K \in \{0,1\}^\lambda$*. It can be computed by a uniform circuit of total size* $T \cdot \mathsf{poly}(\lambda)$ *and depth* $\mathsf{poly}(\log T, \lambda)$*.*
- $(Z, s) \leftarrow \mathsf{Gen}(s, K)$ *is a probabilistic algorithm that takes as input a solution* $s \in \{0,1\}^\lambda$ *and secret key* $K$ *and outputs a puzzle* $Z$*. It can be computed in (sequential) time* $\mathsf{poly}(\log T, \lambda)$*.*
- $s \leftarrow \mathsf{Solve}(P, Z)$ *is a deterministic algorithm that takes as input a state* $P$ *and puzzle* $Z$ *and outputs a solution* $s$*. It can be computed in time* $T \cdot \mathsf{poly}(\lambda)$*.*

*A time-lock puzzle with pre-processing is* correct *if for all* $\lambda$*, for all polynomials* $T$ *in* $\lambda$ *and solution* $s \in \{0,1\}^\lambda$ *it holds:*

$$\Pr\left[s = s' : (P, K) \leftarrow \mathsf{Preproc}(T), Z \leftarrow \mathsf{Gen}(s, K), s' \leftarrow \mathsf{Solve}(P, Z)\right] = 1.$$

*Remark 1.* We note that it is straightforward to adapt our definition of TLPs to ones with pre-processing. As this will not have an impact of any of our constructions that use TLPs, we will not make this explicit henceforth.

$\underline{\mathsf{ExpTLP}^b_{\mathcal{A}}(1^\lambda)\text{:}}$
$(Z, s) \leftarrow \mathsf{Gen}(T(\lambda)), b \xleftarrow{\$} \{0,1\}$
if $b = 0 : c := s$
if $b = 1 : c \xleftarrow{\$} S$
return $b' \leftarrow \mathcal{A}_\lambda(Z, c)$

**Fig. 1.** Security experiment for time lock puzzles.

### 2.2 Incremental Time-Lock Puzzles

We introduce *incremental time-lock puzzles* as a particularly useful generalization of basic time-lock puzzles, which yields particularly practical time-lock encryption schemes. To this end, we generalize Definition 1 by allowing the $\mathsf{Gen}$ algorithm to take multiple different time parameters as input. Algorithm $\mathsf{Gen}$ produces a corresponding set of puzzles.

**Definition 5.** *An* incremental time-lock puzzle *is tuple of algorithms* $\mathtt{TLP} = (\mathsf{Gen}, \mathsf{Solve})$ *with the following syntax.*

- $(Z_i, s_i)_{i\in[n]} \leftarrow \mathsf{Gen}((T_i)_{i\in[n]})$ *is a probabilistic algorithm which takes as input* $n$ *integers* $(T_i)_{i\in[n]}$ *and outputs* $n$ *puzzles together with their solutions* $(Z_i, s_i)_{i\in[n]}$ *in time at most* $\mathsf{poly}((\log T_i)_{i\in[n]}, \lambda)$*. Without loss of generality we assume in the sequel that set* $(T_i)_{i\in[n]}$ *is ordered and hence* $T_i < T_{i+1}$ *for all* $i \in [n-1]$*.*
- $s_i \leftarrow \mathsf{Solve}(Z_i, s_{i-1})$ *is a deterministic algorithm which takes as input a puzzle* $Z_i$ *and a solution for puzzle* $Z_{i-1}$ *and outputs a solution* $s_i$*, where we define* $s_0 := \perp$*. We require that* $\mathsf{Solve}$ *runs in time at most* $(T_i - T_{i-1}) \cdot \mathsf{poly}(\lambda)$*, where we define* $T_0 := 0$*.*

*We say incremental time-lock puzzle is* correct *if for all* $\lambda, n \in \mathbb{N}$*, for all* $i \in [n]$ *and for polynomials* $T_i$ *in* $\lambda$ *such that* $T_i < T_{i+1}$ *it holds:*

$$\Pr\left[s_i = s'_i : (Z_i, s_i)_{i\in[n]} \leftarrow \mathsf{Gen}((T_i)_{i\in[n]}), s'_i \leftarrow \mathsf{Solve}(Z_i, s_{i-1})\right] = 1.$$

*Security.* In order to define a security notion for incremental time-lock puzzle that is useful for our application of constructing particularly efficient timed-release encryption schemes, we need to introduce an additional function $\mathsf{F} : S \to Y$ which takes as input elements of $S$ and outputs elements of some set $Y$. Instead of requiring that elements $s_i$ are indistinguishable from random, we require that $y_i = \mathsf{F}(s_i)$ is indistinguishable from random. We explain the necessity for function $\mathsf{F}$ after the following definition.

**Definition 6.** *Consider the security experiment* $\mathsf{ExpiTLP}^b_{\mathcal{A}}(1^\lambda)$ *in Figure 2. We say that an incremental time lock puzzle* $\mathtt{TLP}$ *is* secure with respect to function $\mathsf{F}$ and with gap $\epsilon < 1$*, if there exists a polynomial* $\tilde{T}(\cdot)$ *such that for all* $n \in \mathbb{N}$*, for all* $i \in [n]$*, for all polynomials* $T_i$ *such that* $T_i(\cdot) \geq \tilde{T}(\cdot)$ *and every polynomial-size*

$\underline{\mathsf{ExpiTLP}^b_{\mathcal{A}}(1^\lambda)\text{:}}$
$(i, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_{1,i}(1^\lambda)$; $b \xleftarrow{\$} \{0,1\}$
$(Z_j, s_j)_{j\in[n]} \leftarrow \mathsf{Gen}((T_j)_{j\in[n]})$
$(y_j := \mathsf{F}(s_j))_{j\in[i-1]}$
if $b = 0 : \forall j \in (i, \ldots, n) : y_j := \mathsf{F}(s_j)$
if $b = 1 : \forall j \in (i, \ldots, n) : y_j \xleftarrow{\$} Y$
return $b' \leftarrow \mathcal{A}_{2,i}((Z_j, y_j)_{j\in[n]}, \mathsf{st})$

**Fig. 2.** Security experiment for incremental time lock puzzles.

*adversary* $\mathcal{A} = \{(\mathcal{A}_{1,i}, \mathcal{A}_{2,i})_\lambda\}_{\lambda\in\mathbb{N}}$ *where the depth of* $\mathcal{A}_{2,i}$ *is bounded from above by* $T_i^\epsilon(\lambda)$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *it holds*

$$\mathbf{Adv}^{\mathsf{TLP}}_{\mathcal{A}} = \left|\Pr\left[\mathsf{ExpiTLP}^0_{\mathcal{A}}(1^\lambda)\right] - \Pr\left[\mathsf{ExpiTLP}^1_{\mathcal{A}}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda).$$

*On the need for function* $\mathsf{F}$. The introduction of function $\mathsf{F}$ in our definition is new and does not appear in prior work. Let us explain why function $\mathsf{F}$ is necessary to achieve security.

In Section 3.3 we will build an incremental timed-release *encryption* scheme, where security is based on the security of an incremental time-lock puzzle. We will consider this scheme insecure, if an adversary runs in time $T < T_i$ and is able to break the security of an encryption with respect to time slot $T_i$ (see Definition 11 for a precise definition). For such an adversary, we need to simulate all values up to $T_{i-1}$, in particular all time lock puzzle solutions $s_1, \ldots, s_{i-1}$ up to $T_{i-1}$, properly, as otherwise the reduction would not simulate the security experiment properly for an adversary running in time $T = T_{i-1} < T_i$, for instance.

Note that we cannot build a reduction which receives as input $s_1, \ldots, s_{i-1}$ as part of the time-lock puzzle instance, because then the reduction would only be able break the assumption that the puzzle is hard if it runs in time less than $T_i - T_{i-1}$. However, the considered running time $T$ of the adversary is only guaranteed to be less than $T_i$, so we cannot achieve any security if $T_i > T \geq T_i - T_{i-1}$. This would hold, however, impose a minimal distance $T_i > 2T_{i-1}$ between each two consecutive time slots, which we consider as very undesirable and impractical for applications. We would rather allow a constant distance $d$ such that $T_i = T_{i-1} + d$.

Note that we also cannot build a reduction which computes $s_1, \ldots, s_{i-1}$ itself, since then the running time of the reduction *without* the adversary would already be $T_{i-1}$, such that together with the running time $T$ of the adversary we would have a total running time of the reduction of $T_{i-1} + T$, which is again too large to yield a meaningful reduction if $T_{i-1} + T > T_i$.

Our solution to overcome this difficulty is to construct a timed-release encryption scheme which does not directly use the real solutions $s_i$, but instead $\mathsf{F}(s_i)$ where one can think of $\mathsf{F}$ as a hard-to-invert function. This way we are able to formulate a hardness assumption for time-lock puzzles where the re-

duction in the security proof of the timed-release encryption scheme) receives $\mathsf{F}(s_1), \ldots, \mathsf{F}(s_{i-1})$ as additional "advice", and thus is able to provide a proper simulation. At the same time it is reasonable to assume that no adversary (here: our reduction) is able to distinguish $\mathsf{F}(s_i)$ from random, even if it runs in time up to $T < T_i$, which is exactly the upper bound that we have on the timed-release encryption adversary.

The work of Malavolta and Thyagarajan [MT19, Section 5.2] also proposed a construction that allows to use multiple time slots, by describing a specific construction which is similar to our notion of incremental timed-release encryption. The technical difficulty described here should arise in their construction as well. Unfortunately, they do not provide a formal security analysis, so that this is not clarified. We believe that a similar assumption involving an "advice" for the reduction is also necessary for a security proof of this construction suggested in their work.

### 2.3 Instantiating (Incremental) TLPs from Sequential Squaring

Subsequently, we discuss instantiations of TLPs based on the sequential squaring assumption. Therefore, we recall a definition of the sequential squaring assumption which was implicitly introduced by Rivest et al. [RSW96]. Let $p$ be an odd prime number. We say that $p$ is a strong prime, if $p = 2p' + 1$ for some prime number $p'$. Let $\mathsf{GenModulus}$ be a probabilistic polynomial-time algorithm which, on input $1^\lambda$, outputs two $\lambda$-bit strong primes $p$ and $q$ and modulus $N$ that is the product of $p$ and $q$. We denote by $\mathbb{J}_N$ the cyclic subgroup of elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$.

**Definition 7 (Sequential Squaring Assumption).** *Let $T(\cdot)$ be a polynomial. The Sequential Squaring Assumptions holds relative to $\mathsf{GenModulus}$ if there exists some $0 < \epsilon < 1$ such that for every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of depth bounded from above by $T^\epsilon(\lambda)$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ it holds*

$$\Pr\left[\begin{array}{rl} & (p,q,N) \leftarrow \mathsf{GenModulus}(1^\lambda) \\ & x \xleftarrow{\$} \mathbb{J}_N, b \xleftarrow{\$} \{0,1\} \\ b = b' \ : \ & \textit{if } b = 0 : y := x^{2^{T(\lambda)}} \mod N \\ & \textit{if } b = 1 : y \xleftarrow{\$} \mathbb{J}_N \\ & b' \leftarrow \mathcal{A}_\lambda(N, T(\lambda), x, y) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

The instantiation of TLP from the sequential squaring assumption is straightforward:

- $\mathsf{Gen}(T)$: Run $(p,q,N) \leftarrow \mathsf{GenModulus}(1^\lambda)$. Randomly sample $x \xleftarrow{\$} \mathbb{J}_N$ and compute the value $s := x^{2^T} \mod N$. Notice that value $s$ can be efficiently computed knowing the values $p$ and $q$. Set $Z := (N, T, x)$ and output $(Z, s)$.
- $\mathsf{Solve}(Z)$: compute $s := x^{2^T} \mod N$ by repeated squaring.

The security of this construction is directly implied by the security of the sequential squaring assumption.

In order to obtain an incremental time-lock puzzle, we propose the following instantiation:

- $\mathsf{Gen}((T_i)_{i\in[n]})$: Run $(p, q, N) \leftarrow \mathsf{GenModulus}(1^\lambda)$. Randomly sample $x \xleftarrow{\$} \mathbb{J}_N$ and compute values $s_i := x^{2^{T_i}} \mod N$ for all $i \in \{1, \ldots, n\}$. Output $((N, x, T_i, T_{i-1}), s_i))_{i\in[n]}$. Value $s_i$ can be efficiently computed knowing the values $p$ and $q$.
- $\mathsf{Solve}((N, x, T_i, T_{i-1}), s_{i-1})$: Compute value $s_{i-1}^{T_i - T_{i-1}} \mod N$ by repeated squaring.
- $\mathsf{F}(s_i)$: return $F(s_i)$.

Our assumptions is that combination of the time lock-puzzle with function $F$ is a secure incremental time-lock puzzle in the sense of Definition 6. We think, that suitable candidate for function $F$ is for example SHA-3.

### 2.4 Instantiating TLPs from Randomized Encodings

Subsequently, we discuss instantiations of TLPs based on and different variants of randomized encodings [IK00,AIK06] and in particular the approach of constructing TLPs from them by Bitansky et al. [BGJ+16].

We first recall TLPs from randomized encodings (REs) in [BGJ+16] and show how to cast them into our TLP framework to obtain secure TLPs according to Definition 1. Subsequently, we focus on constructions of TLPs from standard assumptions and in particular one-way functions (yielding so called weak TLPs) as well as the sub-exponential Learning with Errors (LWE) problem (yielding so called TLPs with pre-processing). Although we omit it here, we note that we could also realize TLPs with the efficiency as in Definition 1 when relying on succinct REs which can be constructed assuming one-way functions and indistinguishability obfuscation (cf. [KLW15]).

First, we recall a TLP $\mathtt{TLP}' = (\mathsf{Gen}', \mathsf{Solve}')$ as defined in [BGJ+16], where the difference to Definition 1 is that the puzzle generation is defined as $Z \leftarrow \mathsf{Gen}'(T, s)$, i.e., the generation of the puzzle already takes it solution $s$. Observe, however, that any such TLP can easily be modified to meet our definition in that $\mathsf{Gen}(T)$ simply internally samples $s \xleftarrow{\$} S$ and then runs $\mathsf{Gen}'(T, s)$ and $\mathsf{Solve}' = \mathsf{Solve}$. We note that this can essentially be viewed as the trivial construction of obtaining a KEM from a public key encryption (PKE) scheme. Consequently, the security of our TLP when based on the one from [BGJ+16] (where the adversary outputs two solutions $(s_0, s_1)$ and obtains a puzzle for one of them) can be argued analogously to how arguing security for the KEM from PKE construction.

**Randomized encodings.** Now, we recall the notion of (reusable) randomized encodings.

**Definition 8 (Randomized Encoding [BGJ+16]).** *A randomized encoding scheme consists of two algorithms* $\mathtt{RE} = (\mathsf{Encode}, \mathsf{Decode})$ *satisfying the following requirements:*

- $\widehat{M}(x) \leftarrow \mathsf{Encode}(M, x, T)$ *is a probabilistic algorithm that takes as input a machine* $M$*, input* $x$ *and time bound* $T$*. The algorithm outputs a randomized encoding* $\widehat{M}(x)$*. It can be computed by a uniform circuit of depth* $\mathsf{polylog}(T) \cdot \mathsf{poly}(|M|, |x|, \lambda)$ *and total size* $T \cdot \mathsf{poly}(|M|, \lambda)$*.*
- $y \leftarrow \mathsf{Decode}(\widehat{M}(x))$ *is a deterministic algorithm that takes as input a randomized encoding* $\widehat{M}(x)$ *and computes an output* $y \in \{0,1\}^\lambda$*. It can be computed in (sequential) time* $T \cdot \mathsf{poly}(|M|, |x|, \lambda)$*.*

For correctness and security we refer to [BGJ+16]. Using the fact that garbled circuits yield randomized encodings (cf. e.g., for discussion [App17]), we have the following:

**Corollary 1.** *Assuming one-way functions, there exists a randomized encoding scheme.*

**Definition 9 (Reusable Randomized Encoding [BGJ+16]).** *A reusable randomized encoding scheme consists of algorithms* $\mathtt{RE} = (\mathsf{Preproc}, \mathsf{Encode}, \mathsf{Decode})$ *satisfying the following requirements:*

- $(\widehat{U}, K) \leftarrow \mathsf{Preproc}(m, n, T)$ *is a probabilistic algorithm that takes as input bounds* $m$*,* $n$*,* $T$ *on machine size, input size, and time. It outputs an encoded state* $\widehat{U}$ *and a short secret key* $K \in \{0,1\}^\lambda$*. It can be computed by a uniform circuit of depth* $\mathsf{polylog}(T) \cdot \mathsf{poly}(m, n, \lambda)$ *and total size* $T \cdot \mathsf{poly}(m, \lambda)$*.*
- $\widehat{M}(x) \leftarrow \mathsf{Encode}(M, x, K)$ *is a probabilistic algorithm that takes as input a machine* $M$*, input* $x$*, secret key* $K \in \{0,1\}^\lambda$ *and outputs a randomized encoding* $\widehat{M}(x)$*. It can be computed in sequential time* $\mathsf{polylog}(T) \cdot \mathsf{poly}(m, n, \lambda)$*.*
- $y \leftarrow \mathsf{Decode}(\widehat{U}, \widehat{M}(x))$ *is a deterministic algorithm that takes as input an encoded state* $\widehat{U}$ *and a randomized encoding* $\widehat{M}(x)$ *and computes an output* $y \in \{0,1\}^\lambda$*. It can be computed in (sequential) time* $T \cdot \mathsf{poly}(m, n, \lambda)$*.*

For correctness and security we refer to [BGJ+16].

**Theorem 1 ([GKP+13]).** *Assuming sub-exponential hardness of the LWE problem, there exists a reusable randomized encoding scheme.*

**TLPs from Randomized Encodings.** Finally, we discuss the construction of wTLPs and ppTLPs from randomized encodings. For wTLPs, let $\mathtt{RE}$ be a randomized encoding scheme. For $s \in \{0,1\}^\lambda$ and $T \leq 2^\lambda$, let $M_s^T$ be a machine that, on any input $x \in \{0,1\}^\lambda$ outputs the string $s$ after $T$ steps. Furthermore, $M_s^T$ is described by $3\lambda$ bits (which is possible for large enough $\lambda$). Then the (w)TLP is constructed as follows:

- $\mathsf{Gen}(T, s)$ : sample $\widehat{M_s^T}(0^\lambda) \leftarrow \mathtt{RE}.\mathsf{Encode}(M_s^T, 0^\lambda, T)$ and output $Z = \widehat{M_s^T}(0^\lambda)$.

– $\mathsf{Solve}(Z)$ : return $\mathtt{RE}.\mathsf{Decode}(Z)$.

**Theorem 2 (Thm 3.10 [BGJ+16]).** *Let $\epsilon < 1$. Assume that, for every polynomial bounded function $T(\cdot)$, there exists a non-parallelizing language $L \in Dtime(T(\cdot))$ with gap $\epsilon$. Then, for any $\epsilon' < \epsilon$, the above construction is a weak time-lock puzzle with gap $\epsilon'$.*

For ppTLPs, the construction is as follows:

– $\mathsf{Preproc}(T)$ : sample $(\widehat{U}, K') \leftarrow \mathtt{RE}.\mathsf{Preproc}(3\lambda, \lambda, T)$ and return $(P = \widehat{U}, K = K')$.
– $\mathsf{Gen}(T, s)$ : sample $\widehat{M_s^T}(0^\lambda) \leftarrow \mathtt{RE}.\mathsf{Encode}(M_s^T, 0^\lambda, K)$ and output $Z = \widehat{M_s^T}(0^\lambda)$.
– $\mathsf{Solve}(P, Z)$ : return $\mathtt{RE}.\mathsf{Decode}(P, Z)$.

For the construction we have the following:

**Theorem 3 (Thm 4.8 [BGJ+16]).** *Let $\epsilon < 1$. Assume that, for every polynomial bounded function $T(\cdot)$, there exists a non-parallelizing language $L \in Dtime(T(\cdot))$ with gap $\epsilon$. Then, for any $\epsilon' < \epsilon$, the above construction is a time-lock puzzle with pre-processing with gap $\epsilon'$.*

*Remark 2.* We mention here that Malavolta and Thyagarajan [MT19] stress that for certain applications (e.g., e-voting or sealed bid auctions) it might be perfectly acceptable to an expensive setup ahead of time to run the parameters such that the time required to solve puzzles start from the moment the setup is finished.

## 3 Generic Constructions of Timed-Release Encryption

In this section we will give generic constructions of (incremental) timed-release encryption schemes based on (incremental) time-lock puzzles. There exist several definitions for a timed-release encryption scheme, we base ours on that of Unruh [Unr14]. However, we introduce two additional algorithms $\mathsf{Setup}$ and $\mathsf{Solve}$ which leads to better modularity and applicability of timed-release encryption scheme, as we will illustrate in Section 4.

**Definition 10.** *An incremental timed-release encryption is tuple of algorithms $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ with the following syntax.*

– $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]} \leftarrow \mathsf{Setup}(1^\lambda, (T_i)_{i\in[n]})$ *is a probabilistic algorithm which takes as input a security parameter $1^\lambda$ and a set of time hardness parameters $(T_i)_{i\in[n]}$ with $T_i < T_{i+1}$ for all $i \in [n-1]$, and outputs set of public encryption parameters and public decryption parameters $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]}$. We require that* $\mathsf{Setup}$ *runs in time* $\mathsf{poly}((\log T_i)_{i\in[n]}, \lambda)$.
– $s_i \leftarrow \mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})$ *is a deterministic algorithm which takes as input public decryption parameters $\mathsf{pp}_{d,i}$ and a solution from a previous iteration $s_{i-1}$, where $s_0 := \perp$, and outputs a solution $s_i$. We require that* $\mathsf{Solve}$ *runs in time at most $(T_i - T_{i-1}) \cdot \mathsf{poly}(\lambda)$.*

– $c \leftarrow \mathsf{Enc}(\mathsf{pp}_{e,i}, m)$ *is a probabilistic algorithm that takes as input public encryption parameters* $\mathsf{pp}_{e,i}$ *and message* $m$*, and outputs a ciphertext* $c$*.*
– $m \leftarrow \mathsf{Dec}(s_i, c)$ *is a deterministic algorithm which takes as input a solution* $s_i$ *and a ciphertext* $c$*, and outputs* $m$*.*

*We say timed-release encryption with multiple time slots is* correct *if for all* $\lambda, n \in \mathbb{N}$*, for all* $i \in [n]$ *and for polynomials* $T_i$ *in* $\lambda$ *such that* $T_i < T_{i+1}$ *and all set of messages* $m$ *it holds:*

$$\Pr\left[m = m' \ : \begin{array}{r}(\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]} \leftarrow \mathsf{Setup}(1^\lambda, (T_j)_{j\in[n]}) \\ s_i \leftarrow \mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1}) \\ m' \leftarrow \mathsf{Dec}(s_i, \mathsf{Enc}(\mathsf{pp}_{e,i}, m_i))\end{array}\right] = 1.$$

Note that the above definition also defines "non-incremental" timed-release encryption, by setting $n = 1$.

**Definition 11.** *An incremental timed-release encryption is* secure with gap $\epsilon < 1$ *if there exists a polynomial* $\tilde{T}(\cdot)$ *such that for all polynomials* $n$ *in* $\lambda$*, for all* $i \in [n]$*, for all polynomials* $T_i$ *such that* $T_i(\cdot) \geq \tilde{T}(\cdot)$ *and every adversary* $\{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *it holds*

$$\mathbf{Adv}^{\mathtt{TRE}}_{\mathcal{A}} = \left|\Pr\left[b = b' \ : \begin{array}{r}\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, (T_j)_{j\in[n]}) \\ (i, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{PP}) \\ b \xleftarrow{\$} \{0,1\};\ c \leftarrow \mathsf{Enc}(\mathsf{pp}_{e,i}, m_b) \\ b' \leftarrow \mathcal{A}_2(c, \mathsf{st})\end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

*We require that an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *where* $\mathcal{A}_1$ *outputs* $i$ *in the second step of the above security experiment consists of two circuits with* total *depth at most* $T^\epsilon_i(\lambda)$ *(i. e., the total depth is the sum of the depth of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$*).*

### 3.1 Basic Construction

**Building blocks.** Our construction combines a time-lock puzzle with a standard public-key encryption scheme.

**Definition 12.** *A public key encryption scheme* $\mathtt{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is triple of efficient algorithms.*

– $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ *is a probabilistic algorithm which on input* $1^\lambda$ *outputs a public/secret key pair.*
– $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ *is a probabilistic algorithm that takes as input a public key* $\mathsf{pk}$ *and a message* $m$ *and outputs a ciphertext* $c$*.*
– $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$ *is a deterministic algorithm which on input a secret key* $\mathsf{sk}$ *and a ciphertext* $c$ *outputs* $m \in \mathcal{M} \cup \{\perp\}$*.*

*We say* PKE *is correct if for all* $\lambda \in \mathbb{N}$ *and all* $m \in \mathcal{M}$ *holds:*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)] = 1$$

We require standard IND-CPA security of the public-key encryption scheme, since this is sufficient to construct a timed-release encryption scheme achieving Definition 11. A stronger "CCA-style" security notion for timed-release encryption would be achievable by replacing the below definition with IND-CCA security. However, we consider this as not very useful for timed-release encryption, since it is unclear where in an application a "CCA-oracle" could plausibly exist in an application *before* the release time is reached, since the decryption key is hidden until this point in time. After the release time the ciphertext will be decryptable, anyway, so we have no security expectations. However, some applications may require non-malleability of ciphertexts, which could be achieved via an IND-CCA secure public-key encryption scheme, for instance.

**Definition 13.** *A* PKE *scheme is* secure *if for all PPT adversaries* $\mathcal{A}$ *there is a negligible function* $\mathsf{negl}$ *such that*

$$\mathbf{Adv}^{\mathtt{PKE}}_{\mathcal{A}} = \left| \Pr\left[ b = b' \ : \begin{array}{r} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, \mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\};\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}(c) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

**Construction.** Let $\mathtt{TLP} = (\mathtt{TLP.Gen}, \mathtt{TLP.Solve})$ be a time-lock puzzle with solution space $S$ and let $\mathtt{PKE} = (\mathtt{PKE.Gen}, \mathtt{PKE.Enc}, \mathtt{PKE.Dec})$ be a public key encryption scheme. Figure 3 describes our construction of a timed-release encryption scheme. Observe that correctness is directly implied by correctness of the public key encryption scheme and the time-lock puzzle.

| | |
|---|---|
| $\underline{\mathsf{Setup}(1^\lambda, T)}$ | $\underline{\mathsf{Solve}(\mathsf{pp}_d)}$ |
| $(Z, s) \leftarrow \mathtt{TLP.Gen}(T)$ | $s \leftarrow \mathtt{TLP.Solve}(\mathsf{pp}_d)$ |
| $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE.Gen}(1^\lambda; s)$ | return $s$ |
| return $\mathsf{pp}_e := \mathsf{pk}, \mathsf{pp}_d = Z$ | |
| | |
| $\underline{\mathsf{Enc}(\mathsf{pp}_e, m)}$ | $\underline{\mathsf{Dec}(s, c)}$ |
| return $c \leftarrow \mathtt{PKE.Enc}(\mathsf{pp}_e, m)$ | $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE.Gen}(1^\lambda; s)$ |
| | return $m \leftarrow \mathtt{PKE.Dec}(\mathsf{sk}, c)$ |

**Fig. 3.** Construction of TRE

**Theorem 4.** *If* $\mathtt{TLP} = (\mathtt{TLP.Gen}, \mathtt{TLP.Solve})$ *is secure time-lock puzzle with gap* $\epsilon$ *in the sense of Definition 2 and* $\mathtt{PKE} = (\mathtt{PKE.Gen}, \mathtt{PKE.Enc}, \mathtt{PKE.Dec})$ *is a CPA secure encryption scheme according to Definition 13, then* $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Solve}, \mathsf{Enc},$

$\underline{\mathsf{Setup}'(1^\lambda, T)}$
$(Z, s) \leftarrow \mathtt{TLP.Gen}(T)$; $s' \xleftarrow{\$} S$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE.Gen}(1^\lambda; s')$
return $\mathsf{pp}_e := \mathsf{pk}, \mathsf{pp}_d = Z$

**Fig. 4.** Alternative setup

Dec) *defined in Figure 3 is a secure timed-release encryption scheme with gap* $\underline{\epsilon} < \epsilon$ *in the sense of Definition 11.*

*Proof.* To prove security we define two games $\mathsf{G}_0$ and $\mathsf{G}_1$ and show that these are computationally indistinguishable.

*Game 0.* Game $\mathsf{G}_0$ corresponds to original security experiment, where we use the Setup directly from our construction.

*Game 1.* In game $\mathsf{G}_1$ we replace Setup with the alternative setup algorithm $\mathsf{Setup}'$ from Figure 4, in which we sample $s'$ independently at random from $S$ and use it as randomness for PKE.Gen.

Let $\mathsf{poly}_{\mathtt{PKE}}(\lambda)$ be the fixed polynomial which bounds the time required to run PKE.Gen and PKE.Enc. Set $\underline{T} := (\mathsf{poly}_{\mathtt{PKE}}(\lambda))^{1/\underline{\epsilon}}$.

**Lemma 1.** *From any adversary* $\mathcal{A} = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ *whose depth is bounded from above by* $T^{\underline{\epsilon}}(\lambda)$ *for some* $T(\cdot) \geq \underline{T}(\cdot)$ *we can construct an algorithm* $\mathcal{B}$ *whose depth is bounded by* $T^{\epsilon}(\lambda)$ *with*

$$\mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}} \geq |\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]|$$

To prove this claim we construct an adversary $\mathcal{B}$ as follows.

1. $\mathcal{B}$ receives $(Z, s)$ and begins to simulate the security experiment from Definition 11 by generating $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE.Gen}(1^\lambda, s)$.
2. Then it runs $\mathcal{A}_1$ which yields $(m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \mathsf{pk}, Z)$.
3. $\mathcal{B}$ picks $b \xleftarrow{\$} \{0, 1\}$ and computes $c \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}, m_b)$.
4. Finally, it runs $b' \leftarrow \mathcal{A}_2(c, \mathsf{st})$ and returns the truth value of $b' = b$.

Note that if $s$ is the solution of the puzzle $Z$, then $\mathcal{B}$ simulates $\mathsf{G}_0$ perfectly. If $s$ is random, then $\mathsf{G}_1$. Moreover, $\mathcal{B}$ meets the depth constraint:

$$\mathsf{depth}(\mathcal{B}(\lambda)) = \mathsf{poly}_{\mathtt{PKE}}(\lambda) + \mathsf{depth}(\mathcal{A}(\lambda)) = \underline{T}^{\underline{\epsilon}}(\lambda) + T^{\underline{\epsilon}}(\lambda) \leq 2T^{\underline{\epsilon}}(\lambda) = o(T^{\underline{\epsilon}}(\lambda)).$$

Thus, we can conclude that $|\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| = \mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}}$ as required.

Now we can show that we can construct an adversary $\mathcal{B}'$ against the PKE scheme.

**Lemma 2.** *From any adversary $\mathcal{A} = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ we can construct an algorithm $\mathcal{B}'$ such that the running time of $\mathcal{B}'$ consists of running $\mathcal{A}$ once plus a small number of additional operations to simulate $\mathsf{G}_1$ such that*

$$\mathbf{Adv}^{\mathtt{PKE}}_{\mathcal{B}'} = \Pr[\mathsf{G}_1 = 1] - \frac{1}{2}$$

The proof of this claim is straightforward. Notice that in $\mathsf{G}_1$ we use fresh randomness which is independent of the puzzle $Z$. We construct $\mathcal{B}'$ as follows:

1. $\mathcal{B}'$ has $\mathsf{pk}$ as input and starts to simulate $\mathsf{G}_1$ by running $(Z, s) \leftarrow \mathtt{TLP.Gen}(T)$.
2. Then it runs adversary $(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pk}, Z)$.
3. It outputs $(m_0, m_1)$ to its experiment and receives $c$.
4. Finally it returns $b \leftarrow \mathcal{A}_2(c, \mathsf{st})$.

Adversary $\mathcal{B}'$ simulates $\mathsf{G}_1$ perfectly. Since $\mathcal{A}$ has polynomially-bounded size, this proves the claim.

By combining Lemma 1 and Lemma 2 we obtain following:

$$\mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}} + \mathbf{Adv}^{\mathtt{PKE}}_{\mathcal{B}'} = |\Pr[\mathsf{G}_0 = 1] - \Pr[\mathsf{G}_1 = 1]| + \left|\Pr[\mathsf{G}_1 = 1] - \frac{1}{2}\right| \geq \mathbf{Adv}^{\mathtt{TRE}}_{\mathcal{A}}$$

which concludes the proof of security.

### 3.2 Discussion

Malavolta et al. [MT19] have introduced *homomorphic* timed-release encryption (called time-lock puzzles in [MT19]), which enables the homomorphic evaluation of functions on encrypted messages. This is extremely useful, because in prior constructions it was necessary to solve one puzzle per ciphertext.

The very clever idea in [MT19] is that the homomorphism makes it possible to first combine many encrypted messages into a single ciphertext, such that then only a single puzzle needs to be solved in order to obtain the result. This *"solve one, get many for free"* property scales well across multiple ciphertexts, since it avoids expensive parallel computations to solve one puzzle per ciphertext, while it still achieves the desired security against time-bounded adversaries.

Note however that the approach is still somewhat limited with respect to applicability. The homomorphic evaluation of ciphertexts is only useful when an application needs to decrypt only a function $f(m_1, \ldots, m_n)$ of all encrypted messages $m_1, \ldots, m_n$. However, if it needs to learn all $n$ messages $m_1, \ldots, m_n$ explicitly, then [MT19] still requires to solve $n$ puzzles in parallel. Also a homomorphic encryption scheme that supports the homomorphic evaluation of function $f$ is required in order to take advantage of the *"solve one, get many"* property. Practically efficient instantiations of additively and multiplicatively homomorphic schemes are readily available, but fully-homomorphic schemes [Gen09] are currently still much less practical. So for applications that require a complex function $f$, the homomorphic approach from [MT19] is conceptually interesting, but not yet practical.

*"Solve one, get many for free" in our construction.* Our construction achieves this "solve one, get many" property as well, even without requiring an underlying homomorphic encryption schem. This is because solving a puzzle yields the secret key of the PKE scheme, which makes it possible to decrypt all ciphertexts efficiently without requiring to solve many puzzles in parallel. This makes it possible to achieve this property even for applications that require the full decryption of all encrypted messages $m_1, \ldots, m_n$, for instance in order to obtain practical schemes that evaluate a complex function $f(m_1, \ldots, m_n)$ after decryption.

Our scheme is the first to achieve an amortized complexity of decryption per ciphertext of

$$\frac{n \cdot T_{\mathsf{PKE.Dec}} + T_{\mathtt{TLP}}}{n} \tag{1}$$

where $T_{\mathsf{PKE.Dec}}$ is the time required to run the decryption algorithm $\mathsf{PKE.Dec}$ and $T_{\mathtt{TLP}}$ is the time required to solve the puzzle. Note that this approaches $T_{\mathsf{PKE.Dec}}$ with increasing $n$.

*Using homomorpic encryption in our construction.* In some applications it may be desirable to enable the homomorphic evaluation of ciphertexts before decryption. This might be useful to save space, since it does not require storage of $n$ encryptions of $m_1, \ldots, m_n$, but only of their homomorphic evaluation.

Note that our modular approach readily supports this in a black-box manner, by simply replacing the PKE scheme with a *homomorphic* PKE scheme that supports homomorphic evaluation of ciphertexts. Since the existence of an additional homomorphic evaluation algorithm is merely an additional *functional* feature of the encryption scheme, the security analysis carries over without any modifications. In particular, note that our construction readily supports any (additively/multiplicatively/fully) homomorphic encryption scheme in a modular way, based on arbitrary hardness assumptions and without introducing further additional requirements, such as the need for indistinguishability obfuscation for the fully-homomorphic construction in [MT19], or the need for a *multi-key* fully-homomorphic encryption scheme [LTV12] as in [BDGM19].

### 3.3 Incremental TRE

Next we construct *incremental* timed-release encryption scheme, which can make timed-release encryption even more useful for practical applications. In the sequel let $\mathtt{TLP} = (\mathsf{TLP.Gen}, \mathsf{TLP.Solve})$ be an incremental time-lock puzzle in the sense of Definition 5 and let $\mathtt{PKE} = (\mathsf{PKE.Gen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a public key encryption scheme. Let $\mathsf{F} : S \to Y$ be a function that maps the solution space $S$ of $\mathtt{TLP}$ to the randomness space of algorithm $\mathsf{PKE.Gen}$. Our constructions of an incremental timed-release encryption scheme $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ is given in Figure 5.

Note that correctness of the scheme is directly implied by correctness of the public key encryption scheme and the incremental time-lock puzzle.

| $\underline{\mathsf{Setup}(1^\lambda, (T_i)_{i\in[n]})}$ | $\underline{\mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})}$ |
|---|---|
| $(Z_i, s_i)_{i\in[n]} \leftarrow \mathtt{TLP}.\mathsf{Gen}((T_i)_{i\in[n]})$ | $s_i \leftarrow \mathtt{TLP}.\mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})$ |
| $((\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(s_i)))_{i\in[n]}$ | return $s_i$ |
| return $(\mathsf{pp}_{e,i} := \mathsf{pk}_i, \mathsf{pp}_{d,i} := Z_i)_{i\in[n]}$ | |
| $\underline{\mathsf{Enc}(\mathsf{pp}_{e,i}, m)}$ | $\underline{\mathsf{Dec}(s_i, c)}$ |
| return $c \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, m)$ | $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(s_i))$ |
| | return $m \leftarrow \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}_i, c)$ |

**Fig. 5.** Construction of incremental TRE

**Theorem 5.** *If* $\mathtt{TLP} = (\mathtt{TLP}.\mathsf{Gen}, \mathtt{TLP}.\mathsf{Solve})$ *is a secure incremental time-lock puzzle with gap* $\epsilon$ *w.r.t. function* $\mathsf{F}$ *and* $\mathtt{PKE} = (\mathtt{PKE}.\mathsf{Gen}, \mathtt{PKE}.\mathsf{Enc}, \mathtt{PKE}.\mathsf{Dec})$ *is a CPA secure encryption scheme, then* $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ *defined in Figure 5 is secure incremental timed-release encryption with gap* $\underline{\epsilon} < \epsilon$.

*Proof.* To prove security we define a short sequence of games $\mathsf{G}_0$, $\mathsf{G}_1$, and $\mathsf{G}_2$. Individual games differ in how the setup is realized.

*Game 0.* Game $\mathsf{G}_0$ is the original security experiment with scheme TRE.

*Game 1.* This game is identical to $\mathsf{G}_0$, except that at the beginning of game $\mathsf{G}_1$ we guess an index $i^* \xleftarrow{\$} [n]$ uniformly at random. When $\mathcal{A}_1$ outputs $(i, m_0, m_1, \mathsf{st})$, then we check whether $i = i^*$. If $i \neq i^*$, then we sample and output a random bit $b \xleftarrow{\$} \{0,1\}$ and abort. Otherwise, we continue as in $\mathsf{G}_0$.

**Lemma 3.** *We have*

$$\mathbf{Adv}^{\mathsf{G}_0}_{\mathcal{A}} = n \cdot \mathbf{Adv}^{\mathsf{G}_1}_{\mathcal{A}}$$

This lemma is proven using a standard argument:

$$\begin{aligned}
\Pr[\mathsf{G}_1 = 1] &= \Pr[\mathsf{G}_1 = 1 | i \neq i^*] \Pr[i \neq i^*] + \Pr[\mathsf{G}_1 = 1 | i = i^*] \Pr[i = i^*] \\
&= \frac{1}{2}(1 - \Pr[i = i^*]) + \Pr[\mathsf{G}_1 = 1 | i = i^*] \Pr[i = i^*] \\
&= \Pr[i = i^*](\Pr[\mathsf{G}_1 = 1 | i = i^*] - \frac{1}{2}) + \frac{1}{2} \\
&= \frac{1}{n}(\Pr[\mathsf{G}_0 = 1] - \frac{1}{2}) + \frac{1}{2}
\end{aligned}$$

In the last equality we use the fact output of $\mathcal{A}$ is the same in both experiments if $i = i^*$ and hence $\Pr[\mathsf{G}_1 = 1 | i = i^*] = \Pr[\mathsf{G}_0 = 1]$. Now we can rearrange the terms:

$$\Pr[\mathsf{G}_1 = 1] - \frac{1}{2} = \frac{1}{n}(\Pr[\mathsf{G}_0 = 1] - \frac{1}{2}) \iff \mathbf{Adv}^{\mathsf{G}_1}_{\mathcal{A}} = \frac{1}{n}\mathbf{Adv}^{\mathsf{G}_0}_{\mathcal{A}}$$

*Game 2.* In Game $\mathsf{G}_2$ we replace $\mathsf{Setup}$ with the alternative setup algorithm $\mathsf{Setup}'$ from Figure 6, which takes as input $i^*$. For all $j \in [i^* - 1]$ we produce keys $\mathsf{pk}_j$ using $\mathsf{F}(s_j)$. The remaining keys $\mathsf{pk}_j$ for $j \in [i^*, n]$ are generated using fresh randomness sampled uniformly from the domain of the function $\mathsf{F}$.

$\underline{\mathsf{Setup}'(1^\lambda, (T_j)_{j\in[n]}, i^*)}$
$(Z_j, s_j)_{j\in[n]} \leftarrow \mathtt{TLP.Gen}((T_j)_{j\in[n]})$
$((\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathtt{PKE.Gen}(1^\lambda; \mathsf{F}(s_j)))_{j\in\{[i^*-1]\}}$
$\forall j \in \{i^*, \ldots, n\} : y_j \xleftarrow{\$} Y, (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathtt{PKE.Gen}(1^\lambda; y_j)$
return $(\mathsf{pp}_{e,j} := \mathsf{pk}_j, \mathsf{pp}_{d,j} := Z_j)_{j\in[n]}$

**Fig. 6.** Alternative setup

Let $\mathsf{poly}_{\mathtt{PKE}}(\lambda)$ be the fixed polynomial which bounds the time required to sample random index from the set $[n]$, to run $\mathtt{PKE.Gen}$ $n$-times and to run $\mathtt{PKE.Enc}$. Set $\underline{T} := (\mathsf{poly}_{\mathtt{PKE}}(\lambda))^{1/\epsilon}$.

**Lemma 4.** *From any adversary $\mathcal{A} = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ whose depth is bounded from above by $T^\epsilon(\lambda)$ for some $T(\cdot) \geq \underline{T}(\cdot)$ we can construct an algorithm $\mathcal{B}$ whose depth is bounded by $T^\epsilon(\lambda)$ with*

$$\mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}} \geq |\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]|$$

To prove this claim we construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ as follows.

1. $\mathcal{B}_1$ chooses $i^* \xleftarrow{\$} [n]$, sets $\mathsf{st} := i^*$ and outputs it to its experiment.
2. $\mathcal{B}_2$ receives $(Z_j, y_j)_{j\in[n]}$ and simulates the game by running $((\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathtt{PKE.Gen}(1^\lambda; y_j))_{j\in[n]}$.
3. Then it runs adversary $(i, m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_1(1^\lambda, (\mathsf{pk}_j, Z_j)_{j\in[n]}, \mathsf{st})$.
4. If $i \neq i^*$, then it returns a random bit $b' \xleftarrow{\$} \{0, 1\}$.
5. Otherwise it picks $b \xleftarrow{\$} \{0, 1\}$ and compute $c \leftarrow \mathtt{PKE.Enc}(\mathsf{pk}_i, m_b)$.
6. It runs $b' \leftarrow \mathcal{A}_2(c, \mathsf{st})$ and returns the truth value of $b' = b$.

If $y_j = \mathsf{F}(s_j)$ for all $j \in [n]$, then $\mathcal{B}$ simulates Game $\mathsf{G}_1$ perfectly. If the $y_j$ are random for $j \geq i^*$, then it simulates Game $\mathsf{G}_2$ perfectly. Therefore we obtain

$$\Pr[\mathsf{G}_1 = 1] = \Pr\left[\mathsf{ExpTLP}^0_{\mathcal{B}}(1^\lambda) = 1\right] \text{ and } \Pr[\mathsf{G}_2 = 1] = \Pr\left[\mathsf{ExpTLP}^1_{\mathcal{B}}(1^\lambda) = 1\right]$$

and thus

$$\begin{aligned}\mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}} &\geq \left|\Pr\left[\mathsf{ExpTLP}^0_{\mathcal{B}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{ExpTLP}^1_{\mathcal{B}}(1^\lambda) = 1\right]\right| \\ &= |\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]|\end{aligned}$$

Moreover, $\mathcal{B}$ fulfils the depth constraint:

$$\mathsf{depth}(\mathcal{B}(\lambda)) = \mathsf{poly}_{\mathtt{PKE}}(\lambda) + \mathsf{depth}(\mathcal{A}(\lambda)) = \underline{T}^\epsilon(\lambda) + T^\epsilon(\lambda) \leq 2T^\epsilon(\lambda) = o(T^\epsilon(\lambda)).$$

**Lemma 5.** *From any adversary $\mathcal{A} = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ we can construct an algorithm $\mathcal{B}'$ such that the running time of $\mathcal{B}'$ consists of running $\mathcal{A}$ once plus a small number of additional operations to simulate $\mathsf{G}_2$ such that*

$$\mathbf{Adv}^{\mathtt{PKE}}_{\mathcal{B}'} = \Pr[\mathsf{G}_2 = 1] - \frac{1}{2}$$

The proof is essentially identical to the corresponding step from proof of Theorem 4, adopted to the incremental setting. We construct $\mathcal{B}'$ as follows:

1. $\mathcal{B}'$ receives as input $\mathsf{pk}$ as input and starts to simulate $\mathsf{G}_2$ by sampling $i^* \xleftarrow{\$} [n]$ uniformly random and running $(Z_j, s_j)_{j\in[n]} \leftarrow \mathtt{TLP}.\mathsf{Gen}((T_j)_{j\in[n]})$.
2. For all $j \in [i^* - 1]$ it sets $\mathsf{pk}_j := \mathtt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(s_j))$. The $i^*$-th public key is defined as $\mathsf{pk}_{i^*} := \mathsf{pk}$. For all $j \in [i^* + 1, n]$, the corresponding public key is defined as $\mathsf{pk}_j := \mathtt{PKE}.\mathsf{Gen}(1^\lambda, y_j)$, where $y_j \xleftarrow{\$} Y$.
3. $\mathcal{B}'$ runs adversary $(i, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_1(1^\lambda, (\mathsf{pk}_j, Z_j)_{j\in[n]}, \mathsf{st})$.
4. If $i \neq i^*$ it samples and putpits $b' \xleftarrow{\$} \{0, 1\}$.
5. Else it sends $(m_0, m_1)$ to its experiment and receives $c$.
6. Finally, it returns $b \leftarrow \mathcal{A}_2(c, \mathsf{st})$.

Note that adversary $\mathcal{B}'$ simulates $\mathsf{G}_2$ perfectly, which yields Lemma 5. Finally, combining Lemmas 3-5 we obtain

$$\begin{aligned} n\left(\mathbf{Adv}^{\mathtt{TLP}}_{\mathcal{B}} + \mathbf{Adv}^{\mathtt{PKE}}_{\mathcal{B}'}\right) &= n\left(\left|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]\right| + \left|\Pr[\mathsf{G}_2 = 1] - \frac{1}{2}\right|\right) \\ &\geq n \cdot \mathbf{Adv}^{\mathsf{G}_1}_{\mathcal{A}} = n\left|\Pr[\mathsf{G}_1 = 1] - \frac{1}{2}\right| = \mathbf{Adv}^{\mathtt{TRE}}_{\mathcal{A}} \end{aligned}$$

which concludes the proof of security.

### 3.4 Incremental Timed-Release Encryption with Public Servers

One particularly nice feature of our notion of incremental timed-release encryption is that one can use a *single* centralized server that continuously computes and publishes solutions $s_i = \mathsf{Solve}(s_{i-1})$ to decrypt an arbitrary number of ciphertexts. Most importantly, the server would be *independent* of these ciphertexts, which is not achieved by prior constructions.

This yields timed-release encryption where the decrypting parties would not have to solve any puzzle, but merely would have to wait until the server publishes a solution. Note that here the fact that the amortized complexity of decrypting $n$ ciphertexts approaches the complexity of running $\mathtt{PKE}.\mathsf{Dec}$ with increasing $n$ is particularly useful (cf. Equation (1)).

*Third-party servers and trusted setup.* If one is worried about a trusted setup performed by a third-party server, or about the fact that one server might run out of service, then one could use $N$ servers. The public parameters of each server would be used to encrypt a share of the message, using an $(K, N)$-threshold

secret sharing scheme (e.g., [Sha79]). Even with $K-1$ colluding servers, the message would remain hidden. Even if up to $N-K$ servers go out of service, messages would still be recoverable using the $K$ shares obtained from the remaining servers.

### 3.5 Incremental Homomorphic Timed-Release Encryption

Note our contruction of incremental timed-release encryption also immediately achieves the *"solve one, get many for free"* property described in Section 3.2. Furthermore, with the same arguments as in Section 3.2, it readily supports any (additively/multiplicatively/fully) homomorphic encryption scheme in a modular way. However, note that efficient homomorphic computations with a standard homomorphic encryption scheme are only supported *within* a single time period. Homomorphic computations *across* different time slots require *multi-key* homomorphic encryption scheme [LTV12], as in [BDGM19].

## 4 Applications

In this section, we give two application flavors of our timed-released encryption paradigm. The first one is a cryptographic application related to the public-key encryption concept of functional encryption (FE) [BSW12], where we integrate the timed-release features into FE. The second one is the real-world application towards e-voting, multi-party coin flipping, sealed bid auctions, and multi-party contract signing, where we discuss more efficient variants of protocols described by Malavolta and Thyagarajan in [MT19].

### 4.1 Cryptographic Application: Integrating Timed-Release Features into Functional Encryption

In this section, we provide (incremental) timed-release features for functional encryption (FE) in two flavors. We recap FE and its correctness and security notions in Supplementary Material A, where we also discuss two concrete variants, namely identity-based encryption (IBE) and inner-product functional encryption (IPBE).

**Functional Timed-Release Encryption.** We introduce the notion of a functional timed-release encryption (FTRE) scheme. The basic idea is that in such a scheme like in an $\mathtt{FE}$ scheme there is a public key $\mathsf{pk}$ used for encryption of any message $x$ and a master secret key $\mathsf{sk}$ which is associated to a class of functions $\mathcal{F}: \mathcal{X} \to \mathcal{Y}$. In contrast to an $\mathtt{FE}$ scheme, however, in an $\mathtt{FTRE}$ scheme the master secret key can generate public decryption parameters and decryption keys for subclasses $\mathcal{F}' \subseteq \mathcal{F}$ of functions which are associated to time hardness parameters $(T_i)_{i\in[n]}$. The decryption algorithm takes the $i$-th decryption key $\mathsf{dk}_i$, a function $f \in \mathcal{F}'$ and a ciphertext to message $x$ and outputs $f(x)$.

**Definition 14.** *An FTRE scheme* $\mathtt{FTRE}$ *for functionality* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ *consists of four PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F})$ *is a probabilistic algorithm which on input* $1^\lambda$ *and class of functions* $\mathcal{F}$*, outputs a public key* $\mathsf{pk}$ *and a master secret key* $\mathsf{msk}$*.*
- $(\mathsf{pp}_{d,i}, \mathsf{dk}_i)_{i\in[n]} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathcal{F}', (T_i)_{i\in[n]})$ *is a probabilistic algorithm that takes as input a master secret key* $\mathsf{msk}$*, a class of functions* $\mathcal{F}' \subseteq \mathcal{F}$ *with* $|\mathcal{F}'| = n$ *and time hardness parameters* $(T_i)_{i\in[n]}$*, and outputs public decryption parameters and decryption keys* $(\mathsf{pp}_{d,i}, \mathsf{dk}_i)_{i\in[n]}$*.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ *is a probabilistic algorithm that takes on input* $\mathsf{pk}$ *and* $x \in \mathcal{X}$*, outputs a ciphertext* $c$ *for* $x$*.*
- $s_i \leftarrow \mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})$*, is a deterministic algorithm which on input public decryption parameters* $\mathsf{pp}_{d,i}$*, and solution* $s_{i-1}$*, outputs solution* $s_i$*.*
- $f_i(x') \leftarrow \mathsf{Dec}(\mathsf{dk}_i, s_i, f_i, c)$ *is a deterministic algorithm which on input decryption key* $\mathsf{dk}_i$*, solution* $s_i$*, a function* $f_i \in \mathcal{F}$*, and ciphertext* $c$*, outputs* $f(x') \in \mathcal{Y} \cup \{\perp\}$*.*

*We say an FTRE scheme* $\mathtt{FTRE}$ *is correct if for all* $\lambda, n \in \mathbb{N}$ *and for all* $T$ *that is a polynomial in* $\lambda$*, for any* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$*, for any* $x \in \mathcal{X}$*,* $\mathcal{F}' \subseteq \mathcal{F}$ *with* $|\mathcal{F}'| = n$*, for any* $f_i \leftarrow \mathcal{F}'$*, for all* $i \in [n]$*, it holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{dk}_i, s_i, f_i, c) = f_i(x) \;:\; \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ (\mathsf{pp}_{d,i}, \mathsf{dk}_i)_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathcal{F}', (T_i)_i) \\ s_i \leftarrow \mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1}), s_0 = \perp \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)\end{array}\right] = 1$$

**Definition 15.** *An FTRE scheme* $\mathtt{FTRE}$ *is FTRE-IND-CPA secure (indistinguishable under chosen-plaintext attacks) if for all PPT adversaries* $\mathcal{A}$*, we have*

$$\mathbf{Adv}^{\mathtt{FTRE}}_{\mathcal{A}} = \left|\Pr\left[b = b' \;:\; \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ (x_0, x_1, st) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\}, c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot)}(st, c^*)\end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda),$$

*for negligible function* $\mathsf{negl}$*, where we require that* $\mathcal{A}$ *only queries* $\mathsf{KeyGen}$ *with functions* $\mathcal{F}' = (f_1, \ldots, f_n)$ *such that* $f_j(x_0) = f_j(x_1)$*, for all* $j \in [n]$*.*

In the following, we show how to construct FTRE from incremental timed-released encryption and functional encryption.

*Construction of FTRE.* Let $\mathtt{TRE} = (\mathtt{TRE.Setup}, \mathtt{TRE.Solve}, \mathtt{TRE.Enc}, \mathtt{TRE.Dec})$ be a TRE scheme and $\mathtt{FE} = (\mathtt{FE.Gen}, \mathtt{FE.KeyGen}, \mathtt{FE.Enc}, \mathtt{FE.Dec})$ be an FE scheme. We construct an FTRE scheme $\mathtt{FTRE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as given in Figure 7.

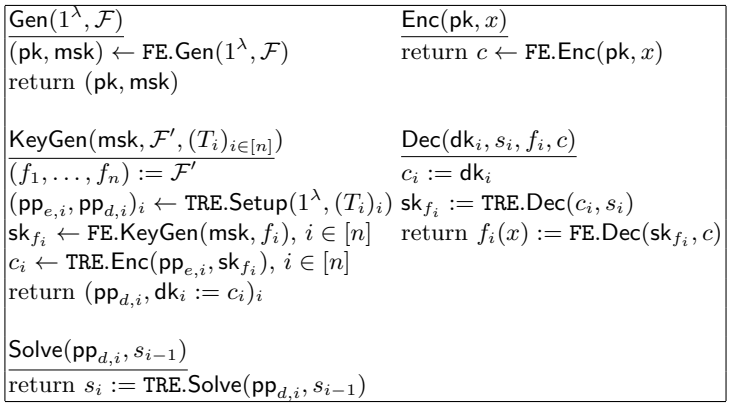


**Fig. 7.** Construction of FTRE.

**Theorem 6.** *If* $\mathtt{TRE} = (\mathtt{TRE}.\mathsf{Setup}, \mathtt{TRE}.\mathsf{Solve}, \mathtt{TRE}.\mathsf{Enc}, \mathtt{TRE}.\mathsf{Dec})$ *is a secure TRE scheme and* $\mathtt{FE} = (\mathtt{FE}.\mathsf{Gen}, \mathtt{FE}.\mathsf{KeyGen}, \mathtt{FE}.\mathsf{Enc}, \mathtt{FE}.\mathsf{Dec})$ *is a FE-IND-CPA secure FE scheme, then* $\mathtt{FTRE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined in Figure 7 is a FTRE-IND-CPA secure FTRE scheme.*

*Proof.* We show how to construct an adversary $\mathcal{B}_{\mathtt{FE}}$ on the FE-IND-CPA security of FE and an adversary $\mathcal{B}_{\mathtt{TRE}}$ on the security of the TRE scheme TRE from an adversary $\mathcal{A}_{\mathtt{FTRE}}$ on the FTRE-IND-CPA security.

We first construct $\mathcal{B}_{\mathtt{FE}}$ as follows:

1. $\mathcal{B}_{\mathtt{FE}}$ receives $\mathsf{pk}$ as input. For any query to $\mathsf{KeyGen}$, $\mathcal{B}_{\mathtt{FE}}$ runs $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_i \leftarrow \mathtt{TRE}.\mathsf{Setup}(1^\lambda, (T_i)_i)$ where the FE key queries are forwarded to its challenger and the results are TRE encrypted under $(\mathsf{pp}_{e,i})_i$.
2. $\mathcal{B}_{\mathtt{FE}}$ runs $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_{\mathtt{FTRE}}(1^\lambda, \mathsf{pk})$.
3. It outputs $(x_0, x_1)$ to its FE challenger and receives $c^*$.
4. Finally, $\mathcal{B}_{\mathtt{FE}}$ sends $b' \leftarrow \mathcal{A}_{\mathtt{FTRE}}(c^*, \mathsf{st})$ to its challenger.

$\mathcal{B}_{\mathtt{FE}}$ simulates the FTRE experiment perfectly. See that if $\mathcal{A}_{\mathtt{FTRE}}$ is a successful adversary, then $\mathcal{B}_{\mathtt{FE}}$ is a successful adversary in the FE-IND-CPA security experiment.

We now construct $\mathcal{B}_{\mathtt{TRE}}$ as follows:

1. $\mathcal{B}_{\mathtt{TRE}}$ receives $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_i$ for time hardness parameters $(T_i)_i$ as input and runs $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$.
2. If $\mathcal{B}_{\mathtt{TRE}}$ is queried on some $T_i$ and function $f_i$, set $f_0 := f_i$ and a uniform function $f_1 \leftarrow \mathcal{F}$. Send $(1, \mathsf{sk}_{f_0}, \mathsf{sk}_{f_1})$ to the challenger and receives $c^*$ where $c^*$ is embedded at the $f_i$-index in the $\mathsf{KeyGen}$-answer.
3. $\mathcal{B}_{\mathtt{TRE}}$ runs $(x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_{\mathtt{FTRE}}(1^\lambda, \mathsf{pk})$ and computes $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b)$, for $b \leftarrow \{0, 1\}$.
4. For $b' \leftarrow \mathcal{A}_{\mathtt{FTRE}}(c, \mathsf{st})$, if $b = b'$, then $\mathcal{B}_{\mathtt{TRE}}$ sends 0 to its challenger, else 1.

$\mathcal{B}_{\mathtt{FE}}$ simulates the FTRE experiment perfectly. See that if $\mathcal{A}_{\mathtt{FTRE}}$ is a successful adversary, then $\mathcal{B}_{\mathtt{TRE}}$ is a successful adversary in the TRE security experiment.

**Application to locked-key IBE.** With FTRE, we are able to lock secret keys of an IBE scheme with a incremental timed-release feature. When the central authority in an IBE scheme generates the identity-based secret keys, it can attach hardness parameters to it such that those keys only become usable incrementally. This, for example, enables an IBE key authority to produce all secret keys in the beginning and afterwards can go offline.

**Timed-Release Functional Encryption.** We define timed-release functional encryption (TRFE) scheme which is notably different to the notion of KTR-FE where the hardness parameter is bound to the key; here, we associate the functional ciphertexts with a hardness parameter.

**Definition 16.** *A TRFE scheme* $\mathsf{TRFE}$ *for functionality* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ *consists of four PPT algorithms* $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}, (T_i)_{i\in[n]})$ *is a probabilistic algorithm which on input* $1^\lambda$*, a class of functions* $\mathcal{F}$*, and hardness parameters* $(T_i)_{i\in[n]}$*, outputs public key* $\mathsf{pk}$ *and a master secret key* $\mathsf{msk}$*.*
- $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ *is a probabilistic algorithm which on input a master secret key* $\mathsf{msk}$ *and a function* $f \in \mathcal{F}$*, outputs secret key* $\mathsf{sk}_f$ *for* $f$*.*
- $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x,\ i)$ *is a probabilistic algorithm which on input* $\mathsf{pk}$*,* $x \in \mathcal{X}$*, and index of the hardness parameter* $i \in [n]$*, outputs a ciphertext* $c_i$ *for* $x$ *associated to index* $i$*.*
- $f(x') \leftarrow \mathsf{Dec}(\mathsf{sk}_f, c_i)$ *is a deterministic algorithm which on input the secret key* $\mathsf{sk}_f$ *and* $c_i$*, outputs* $f(x') \in \mathcal{Y} \cup \{\perp\}$*.*

*We say a TRFE scheme* $\mathsf{TRFE}$ *is correct if for all* $\lambda, n \in \mathbb{N}$ *and for all* $(T_i)_{i\in[n]}$ *that are polynomials in* $\lambda$*, for any* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$*, for any* $x \in \mathcal{X}$*, for any* $f \leftarrow \mathcal{F}$*, for all* $i \in [n]$*, it holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, c_i) = f(x) \ : \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}, (T_i)_{i\in[n])})\\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)\\ c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, x, i)\end{array}\right] = 1$$

**Definition 17.** *A TRFE scheme* $\mathsf{TRFE}$ *is TRFE-IND-CPA secure (indistinguishable under chosen-plaintext attacks) if for all PPT adversaries* $\mathcal{A}$*, we have*

$$\mathbf{Adv}^{\mathsf{TRFE}}_{\mathcal{A}} = \left|\Pr\left[b = b' \ : \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}, (T_i)_{i\in[n])})\\ (x_0, x_1, i^*, st) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{pk})\\ b \leftarrow \{0,1\}, c^*_{i^*} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b, i^*)\\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(st, c^*_{i^*})\end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda),$$

*for negligible function* $\mathsf{negl}$*, where we require that* $\mathcal{A}$ *only queries* $\mathsf{KeyGen}$ *with functions* $f$ *such that* $f(x_0) = f(x_1)$*.*

In the following, we show how to construct TRFE from time-release encryption and functional encryption.

*Construction of TRFE.* Let $\mathtt{TRE} = (\mathtt{TRE.Setup}, \mathtt{TRE.Solve}, \mathtt{TRE.Enc}, \mathtt{TRE.Dec})$ be a TRE scheme and $\mathtt{FE} = (\mathtt{FE.Gen}, \mathtt{FE.KeyGen}, \mathtt{FE.Enc}, \mathtt{E.Dec})$ be an FE scheme. We construct a TRFE scheme $\mathsf{TRFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as given in Figure 8.

| | |
|---|---|
| $\underline{\mathsf{Gen}(1^\lambda, \mathcal{F}, (T_i)_{i\in[n]})}$ | $\underline{\mathsf{Enc}(\mathsf{pk}, x, i')}$ |
| $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]} \leftarrow \mathtt{TRE.Setup}(1^\lambda, (T_i)_{i\in[n]})$ | $((\mathsf{pp}_{e,i})_{i\in[n]}, \mathsf{pk}') := \mathsf{pk}$ |
| $(\mathsf{pk}', \mathsf{msk}') \leftarrow \mathtt{FE.Gen}(1^\lambda, \mathcal{F})$ | $c \leftarrow \mathtt{FE.Enc}(\mathsf{pk}', x)$ |
| $\mathsf{pk} := ((\mathsf{pp}_{e,i})_{i\in[n]}, \mathsf{pk}')$ | return $c_{i'} \leftarrow \mathtt{TRE.Enc}(\mathsf{pp}_{e,i'}, c)$ |
| $\mathsf{msk} := ((\mathsf{pp}_{d,i})_{i\in[n]}, \mathsf{msk}')$ | |
| return $(\mathsf{pk}, \mathsf{msk})$ | |
| | |
| $\underline{\mathsf{KeyGen}(\mathsf{msk}, f)}$ | $\underline{\mathsf{Dec}(\mathsf{sk}_f, c_{i'})}$ |
| $((\mathsf{pp}_{d,i})_{i\in[n]}, \mathsf{msk}') := \mathsf{msk}$ | $((\mathsf{pp}_{d,i})_{i\in[n]}, \mathsf{sk}'_f) := \mathsf{sk}_f, s_0 := \bot$ |
| $\mathsf{sk}'_f \leftarrow \mathtt{FE.KeyGen}(\mathsf{msk}', f)$ | $s_i \leftarrow \mathtt{TRE.Solve}(\mathsf{pp}_{d,i}, s_{i-1}), i \in [i']$ |
| return $\mathsf{sk}_f := ((\mathsf{pp}_{d,i})_{i\in[n]}, \mathsf{sk}'_f)$ | $c \leftarrow \mathtt{TRE.Dec}(c_{i'}, s_{i'})$ |
| | return $f(x) \leftarrow \mathtt{FE.Dec}(\mathsf{sk}_f, c)$ |

**Fig. 8.** Construction of TRFE.

**Theorem 7.** *If* $\mathtt{TRE} = (\mathtt{TRE.Setup}, \mathtt{TRE.Solve}, \mathtt{TRE.Enc}, \mathtt{TRE.Dec})$ *is a secure time-release encryption scheme and* $\mathtt{FE} = (\mathtt{FE.Gen}, \mathtt{FE.KeyGen}, \mathtt{FE.Enc}, \mathtt{FE.Dec})$ *is a FE-IND-CPA secure FE scheme, then* $\mathsf{TRFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined in Figure 8 is a TRFE-IND-CPA secure TRFE scheme.*

*Proof.* We show how to construct an adversary $\mathcal{B}_{\mathtt{FE}}$ on the FE-IND-CPA security of FE and an adversary $\mathcal{B}_{\mathtt{TRE}}$ on the security of the TRE scheme TRE from an adversary $\mathcal{A}_{\mathsf{TRFE}}$ on the TRFE-IND-CPA security.

We first construct $\mathcal{B}_{\mathtt{FE}}$ as follows:

1. $\mathcal{B}_{\mathtt{FE}}$ receives $\mathsf{pk}'$ as input. For any query to KeyGen, $\mathcal{B}_{\mathtt{FE}}$ computes $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]} \leftarrow \mathtt{TRE.Setup}(1^\lambda, (T_i)_{i\in[n]})$ and sets $\mathsf{pk} := (\mathsf{pp}_{e,i}, \mathsf{pk}')$.
2. $\mathcal{B}_{\mathtt{FE}}$ answers KeyGen-queries for $f$ by querying its own challenger KeyGen-oracle with $f$ to receive $\mathsf{sk}'_f$ and returns $\mathsf{sk}_f := ((\mathsf{pp}_{d,i})_{i\in[n]}, \mathsf{sk}'_f)$.
3. $\mathcal{B}_{\mathtt{FE}}$ runs $(x_0, x_1, i^*, \mathsf{st}) \leftarrow \mathcal{A}_{\mathsf{TRFE}}(1^\lambda, \mathsf{pk})$.
4. It outputs $(x_0, x_1)$ to its FE challenger and receives $c$.
5. Finally, $\mathcal{B}_{\mathtt{FE}}$ sends $b' \leftarrow \mathcal{A}_{\mathsf{TRFE}}(\mathsf{st}, c^*)$, for $c^* = \mathtt{TRE.Enc}(\mathsf{pp}_{e,i^*}, c)$ to its challenger.

$\mathcal{B}_{\mathtt{FE}}$ simulates the TRFE experiment perfectly. See that if $\mathcal{A}_{\mathtt{FTRE}}$ is a successful adversary, then $\mathcal{B}_{\mathtt{FE}}$ is a successful adversary in the FE-IND-CPA security experiment.

We now construct $\mathcal{B}_{\mathtt{TRE}}$ as follows:

1. $\mathcal{B}_{\mathtt{TRE}}$ receives $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]}$ as input and runs $(\mathsf{pk}', \mathsf{msk}') \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$. $\mathsf{KeyGen}$-queries are answered using $\mathsf{msk}'$ and $\mathsf{pp}_{d,i}$.
2. $\mathcal{B}_{\mathtt{TRE}}$ runs $(x_0, x_1, i^*, \mathsf{st}) \leftarrow \mathcal{A}_{\mathtt{FTRE}}(1^\lambda, \mathsf{pk})$, for $\mathsf{pk} := ((\mathsf{pp}_{e,i})_{i\in[n]}, \mathsf{pk}')$, and computes $c_0 \leftarrow \mathtt{FE}.\mathsf{Enc}(\mathsf{pk}', x_0)$ and $c_1 \leftarrow \mathtt{FE}.\mathsf{Enc}(\mathsf{pk}', x_1)$. $\mathcal{B}_{\mathtt{TRE}}$ forwards $(i^*, c_0, c_1)$ to its challenger to receive $c^*$.
3. $\mathcal{B}_{\mathtt{TRE}}$ returns $b' \leftarrow \mathcal{A}_{\mathtt{FTRE}}(\mathsf{st}, c^*)$ to its challenger.

$\mathcal{B}_{\mathtt{TRE}}$ simulates the TRFE experiment perfectly. See that if $\mathcal{A}_{\mathsf{TRFE}}$ is a successful adversary, then $\mathcal{B}_{\mathtt{TRE}}$ is a successful adversary in the TRE security experiment.

**Application to inner-product FE (IPFE).** With TRFE, we are able to release the possibility for inner-product function evaluations on ciphertexts incrementally so that they only get available after a certain time has passed. Thus, it is possible to publish data in a way to ensure that statistics about the data can only be computed after some point certain point in time. We note that IPFE is already used for practical applications (e.g., within the FENTEC project[3]).

### 4.2 Real-World Application: Simpler and More Efficient Instantiations

Subsequently, we discuss more efficient variants of protocols which are described in [MT19]. All these protocols require decrypting a set of encrypted messages at some required time. Our approach to TRE allows to decrypt arbitrary number of messages at the specified time by solving one puzzle. In [MT19] this is achieved by homomorphic evaluation of puzzles and then solving one or more resulting puzzles. The drawback of this solution is that one needs to wait until all puzzles of interest have been collected, then execute homomorphic evaluation and only after that the resulting puzzles can be solved. Our scheme allows to start to solve the puzzle immediately after $\mathsf{Setup}$ is run. In all of this applications we are able to use a timed-release encryption scheme without any homomorphic property.

**E-voting.** We focus on designing an e-voting protocol in absence of trusted party, where voters are able to cast their preference without any bias. Similarly to [MT19], we do not consider privacy nor authenticity of the votes. The crucial property of our $\mathtt{TRE}$ is that setup can be reused for producing an arbitrary number of ciphertexts and for that reason it is enough to run $\mathsf{Solve}$ only once. The output $s$ of $\mathsf{Solve}$ allows to obtain the secret key which is then used to decrypt all ciphertexts that have been produced using corresponding $\mathsf{pp}_e$. Therefore, if we encrypt all votes using the same $\mathsf{pp}_e$, we are able to decrypt all ciphertexts at the same time. Then it is easy to obtain final result by combining decrypted plaintexts.

Notice that the security of the timed-release encryption scheme guarantees that all votes remain hidden during the whole voting phase. In the e-voting protocol proposed by Malavolta and Thyagarajan [MT19] we have to wait until

[3] `http://fentec.eu/`

the voting phase is finished and then we can combine puzzles from voting phase to $m$ resulting puzzles. After that these $m$ puzzles can be solved, which requires at least time $T$. Hence, we need time $T$ after the voting phase is over to be able to announce the results. This is in contrast to our protocol in which we can start to solve the puzzle immediately after $\mathsf{Setup}$ is run and hence the results are available at the beginning of the counting phase.

**Multi-Party Coin Flipping.** In multi-party coin flipping we assume $n$ parties which want to flip a coin in the following way: 1) The value of the coin is unbiased even if $n-1$ parties collude and 2) all parties agree on the same value for the coin.

Malavolta and Thyagarajan [MT19] proposed to use their homomorphic time-lock puzzle. Their protocol consist of three phases: Setup, Coin Flipping, Announcement of the Result. Similarly to e-voting protocol, we are able to start solve puzzle only in the last phase and hence obtain results after time $T$. We are able to avoid this problem, by using our timed-release encryption scheme, where we can start to solve the puzzle already after the Setup phase.

**Sealed Bid Auctions.** Here we consider an auction with $n$ bidders. The protocol consist of two phases - the bidding phase and the opening phase. Bids should be kept secret during the bidding phase and later revealed in opening phase. Time-lock puzzles are used in this scenario to mitigate the issue that some bidders can go offline after the bidding phase. If we use only standard time-lock puzzles, then the number of puzzles which has to be solved in opening phase is equal to number of bidders who went offline. In [MT19] this problem was resolved by using homomorphic time-lock puzzles. Their solution has the same issue assume the two previous applications and we can avoid it using the TRE approach.

**Multi-Party Contract Signing.** In multi-party contract signing we assume $n$ parties which want to jointly sign a contract. The parties are mutually distrusting and the contract is valid only if it is signed by all parties. The protocol of Malavolta and Thyagarajan [MT19] consists of four phases - Setup, Key Generation, Signing and Aggregation Phase, and combines RSA-aggregate signatures with multiplicatively homomorphic time-lock puzzles with reusable setup. We remark that time-lock puzzles with reusable setup are in some sense equivalent to our incremental timed-release encryption (though they do only discuss them informally in [MT19]). The protocol runs in $\ell$-rounds. In the $i$-th round every party should create a puzzle with hardness $T_{\ell-i+1}$ which contains a signature of the required message. Hence, the hardness of the puzzles decrease in every round. If some parties haven't broadcasted their puzzles in any round, the parties will homomorphically evaluate puzzles from previous round and solve the resulting puzzle.

Consider a scenario, where in the $i$-th round some party does not broadcast its puzzle. Then if we do not take into account time for homomorphic evaluation we need time $T_{\ell-i+1}$ to solve the resulting puzzle after this event happened. On the other hand, if we use incremental TRE, we are able to obtain result in time

$T_{\ell-i+1}$ after the setup was executed. Moreover, we can combine incremental TRE with arbitrary aggregate signature scheme, because we do not need to perform homomorphic evaluation.

**Acknowledgements.** This work was supported by the German Federal Ministry of Education and Research (BMBF) project REZEIVER, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement n°802823, the EU's Horizon 2020 ECSEL Joint Undertaking under grant agreement n°783119 (SECREDAS) and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (PROFET).

## References

ABDP15. Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 733–751, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

AIK06. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Comput. Complex.*, 15(2):115–162, 2006.

App17. Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In *Tutorials on the Foundations of Cryptography*, pages 1–44. 2017.

BDGM19. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

BF01. Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

BGJ+16. 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: 7th Conference on Innovations in Theoretical Computer Science*, pages 345–356, Cambridge, MA, USA, January 14–16, 2016. Association for Computing Machinery.

BN00. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

BSW12. Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, 2012.

CHKO08. Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11(2):4:1–4:44, 2008.

DN00. Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st Annual Symposium on Foundations of Computer Science*, pages 283–293, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

DOR99. Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 74–89, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

KLW15. Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press.

LJKW18. Jia Liu, Tibor Jager, Saqib A Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, 2018.

LPS17. Huijia Lin, Rafael Pass, and Pratik Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 576–587, Berkeley, CA, USA, October 15–17, 2017. IEEE Computer Society Press.

LTV12. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press.

May93. Timothy C. May. Timed-release crypto. Technical report, February 1993.

MMV11. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany.

MT19. Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

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

Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

Unr14. Dominique Unruh. Revocable quantum timed-release encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 129–146, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

# A Functional Encryption

**Definition 18.** *A functional encryption scheme* $\mathtt{FE}$ *for functionality* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ *consists of four PPT algorithms* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F})$ *is a probabilistic algorithm which on input* $1^\lambda$ *and* $\mathcal{F}$*, outputs a public key and a master secret key.*
- $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$ *is a probabilistic algorithm which on input* $\mathsf{msk}$ *and* $f \in \mathcal{F}$*, outputs a secret key* $\mathsf{sk}_f$ *for* $f$*.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ *is a probabilistic algorithm which on input* $\mathsf{pk}$ *and* $x \in \mathcal{X}$*, outputs a ciphertext* $c$ *for* $x$*.*
- $f(x) \leftarrow \mathsf{Dec}(\mathsf{sk}_f, c)$ *is a deterministic algorithm which on input* $\mathsf{sk}_f$ *and* $c$*, outputs* $f(x) \in \mathcal{Y} \cup \{\perp\}$*.*

*We say an FE scheme* $\mathtt{FE}$ *is correct if for all* $\lambda \in \mathbb{N}$*, for any* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$*, for any* $x \in \mathcal{X}$*, for any* $f \in \mathcal{F}$*, it holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, c) = f(x) \;:\; \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ \mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f) \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)\end{array}\right] = 1$$

**Definition 19.** *An FE scheme* $\mathtt{FE}$ *is FE-IND-CPA secure (indistinguishable under chosen-plaintext attacks) if for all PPT adversaries* $\mathcal{A}$*, we have*

$$\mathbf{Adv}^{\mathtt{FE}}_{\mathcal{A}} = \left|\Pr\left[b = b' \;:\; \begin{array}{r}(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ (x_0, x_1, st) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{pk}) \\ b \leftarrow \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(st, c)\end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda),$$

*for negligible function* $\mathsf{negl}$*, where we require that* $\mathcal{A}$ *only queries* $\mathsf{KeyGen}$ *with functions* $f$ *such that* $f(x_0) = f(x_1)$*.*

**Example: identity-based encryption.** We recall that in identity-based encryption (IBE) messages can be encrypted with respect to any strings as "public keys" (called identities) and decryption requires a secret key for the corresponding identity. Now, an IBE scheme can be obtained from an FE scheme as in Definition 18 by setting the message space $\mathcal{X} := \mathcal{ID} \times \mathcal{M}$ representing pairs of identities and messages $(id, m)$ and $\mathcal{F}$ being an equality testing functionality. A function secret key $\mathsf{sk}_{f_{id^*}}$ for identity $id^*$ is generated with respect to $f_{id^*}$ defined as:

$$f_{id^*}((id, m)) = \begin{cases} m \text{ if } id = id^*, \\ \perp \text{ otherwise.} \end{cases}$$

**Example: inner-product functional encryption.** Inner-product functional encryption (IPFE) allows for computing inner products on encrypted data. The ciphertext in an IPFE is associated to a vector $\boldsymbol{x} = (x_1, \ldots, x_\ell)$ while the secret keys are associated to a function $f_{\boldsymbol{y}}$, for vector $\boldsymbol{y} = (y_1, \ldots, y_\ell)$. An IPFE scheme can be obtained from an FE scheme as in Definition 18 by setting the space $\mathcal{X} := \{(\boldsymbol{x}_i)_i\}$ and $\mathcal{F}$ realizes the inner-product functionality. A function secret key $\mathsf{sk}_{\boldsymbol{y}}$ for vector $\boldsymbol{y}$ is generated with respect to the function $f_{\boldsymbol{y}}$ defined as:

$$f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = x_1 y_1 + \cdots + x_\ell y_\ell.$$