Versatile and Sustainable Timed-Release Encryption and Sequential Time-Lock Puzzles

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 messages "into the future" such that a pre-determined amount of time needs to pass before a message can be accessed. 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 introduced an interesting variant of TRE called homomorphic time-lock puzzles (HTLPs), making TRE more versatile and greatly extending its applications. Here one considers multiple independently generated puzzles and the homomorphic evaluation of a circuit over these puzzles. Solving the so obtained puzzle yields the output of a circuit evaluated on the messages locked by the original puzzles.

We observe that viewing HTLPs more abstractly gives rise to a simple generic construction of homomorphic TRE (HTRE) that is not necessarily based on sequential squaring, but can be instantiated based on any TLP, such as those based on one-way functions and the LWE assumption (via randomized encodings). This construction has slightly different properties, but provides essentially the same functionality for applications. It makes TRE versatile and can be used beyond HTRE, for instance to construct timed-release functional encryption. Interestingly, it achieves a new *"solve one, get many for free"* property, which supports that an arbitrary number of independently time-locked (homomorphically evaluated) messages can all be obtained simultaneously after solving only a single puzzle. This puzzle is independent of the number of time-locked messages and thus achieves optimal amortized cost.

As a second contribution we define and construct *sequential* TLPs as a particularly useful generalization of TLPs and TRE. Such puzzles can be solved sequentially in a way that solving a puzzle additionally considers the previous solution and the time required to solve the puzzle is determined by the difference in the time parameters. When instantiated from sequential squaring, this allows to realize public "sequential squaring services", where everyone can time-lock messages, but only one entity needs to perform the computations required to solve puzzles. Thus, this removes the burden of wasting computational resources by every receiver and makes TRE economically and ecologically more sustainable.

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 [27], 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 (see also [12,11]). In this work we will not focus on this agent-based variant of TRE, but rather on an alternative idea proposed by Rivest *et al.* in [31] and which they 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. The amount of time required to perform this sequential computation is determined by a hardness parameter T 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 hides the message until enough time has elapsed for the puzzle to be solved. TLPs have found numerous applications such as sealed-bid auctions [31], fair contract signing [7], zero-knowledge arguments [13], or non-malleable commitments [22].

The solution proposed by Rivest *et al.* in [31] uses sequential 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 $s = x^{2^T}$ for hardness T is defined as $Z = (N, T, x, x^{2^T} \cdot k, \mathsf{Enc}(k, m))$ where $(x, k) \xleftarrow{\$} (\mathbb{Z}_N^*)^2$ and Enc is a symmetric encryption scheme. Note that this TLP can be equivalently viewed as a timed-release encryption (TRE) that encrypts message m. 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} quickly by taking the shortcut of computing $2^T \bmod \varphi(N)$ directly. 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 [25] 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.* [5] show how to construct TLPs with the aforementioned property from randomized encodings [19,3] relying on indistinguishability obfuscation. Interestingly, when slightly relaxing the requirements and allowing efficient parallel computation in the generation of the puzzles or a solution independent preprocessing (so-called *weak* TLPs), then such TLPs can be constructed generically from one-way functions and directly from the learning with errors (LWE) assumption, respectively, via randomized encodings.

Recently, Malavolta and Thyagarajan [26] (MT19 henceforth) proposed an interesting variant of TLPs called *homomorphic* TLPs (HTLPs). Here one considers multiple puzzles $(Z_1, \ldots, Z_n)$ with hardness parameter T, which can be independently generated by different entities. 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, MT19 also shows 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, or more recently verifiable timed signatures [35], again yielding a number of interesting further applications.

MT19 conjecture that any application that involves a large number of users and thus the constraint of requiring to solve multiple puzzles (in parallel) constitute one of the main obstacles that so far prevented the large scale adoption of TLPs. As already, mentioned, this can be partly mitigated via HTLPs by MT19 and in particular if one is only interested in homomorphic evaluations over multiple messages. We additionally stress that applications requiring to solve multiple puzzles will also represent a huge waste of resources and are thus problematic from an economic and ecological perspective. Moreover, even the requirement for a receiver to only solve a single puzzle on its own may already prevent the application of TRE, e.g., for resource constrained receivers of messages as omnipresent within the Internet of Things (IoT).

1.1 Motivation for our Work

The motivation of our work is twofold. Firstly, our goal is to make TRE more versatile in order to improve on existing applications and to broaden the scope of applications even further. For instance, when we look at the HTLPs in MT19, they construct a linearly homomorphic TLP (LHTLP) from the sequential squaring TLP and Paillier encryption [30]. 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). In order to extend this to fully HTLPs (FHTLPs), MT19 requires sub-exponentially hard indistinguishability obfuscation, whereas in follow-up work Brakerski *et al.* in [9] proposed FHTLPs from standard assumption which itself requires an LHTLP (where to the best of our knowledge the aforementioned is the only known construction). In particular, the idea in [9] is to use a multi-key fully-homomorphic encryption (MK-FHE) scheme [24] to encrypt every message with a fresh key and an LHTP to lock the respective MK-FHE secret keys. To be compatible with the LHTLP, Brakerski *et al.* in particular require a MK-FHE scheme with a linear decryption algorithm.

Unfortunately, all these constructions are not generic as they rely on a single particular construction of an HTLP from sequential squaring (and additionally a very specific MK-FHE scheme in [9]). Moreover, one can observe 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. Although 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 a *"solve one, get many for free"* property. And this even if one wants to obtain *all* encrypted messages in full, instead of only the result of a homomorphic evaluation. Note that, if in contrast to the homomorphically evaluated function over all the messages, one wants to unlock n of the input messages with the approaches in [26,9], it requires to solve n puzzles. Consequently, we ask whether it is possible to come up with an approach that provides the *"solve one, get many for free"* property on an arbitrary number of independently time-locked messages such that it is possible to decrypt all single messages and at the same time. So essentially having a solution that *can be* homomorphic but does not need to be if one is only interested in the single messages and all with only solving a single puzzle. Ideally this approach is generic in nature and thus would allow to construct (homomorphic) timed-release encryption (TRE) generically from *any* TLP.

Secondly, a central drawback of TRE is that it puts considerable computational overhead on the message receiver, i.e., the receiver has to invest lots of computational resources to solve the puzzle to obtain the time-locked message. This makes it undesirable for real-life scenarios from an economic as well as ecological perspective. While HTLPs of MT19 address this problem from a different angle, i.e., homomorphically combine many TLPs such that only one puzzle needs to be solved, this will not reveal the individual messages without solving all puzzles. And while this functionality is a helpful feature in certain applications, it can not be considered a general purpose solution, because the amount of recoverable data is bounded by the amount of data that can be encapsulated in a single ciphertext. Moreover, it still requires the receiver to waste potentially significant resources for solving a new puzzle. Consequently, we ask whether this can be avoided. Looking ahead, we remedy this issue via the notion of sequential TLPs from which one obtains sequential TRE. The basic idea is that the puzzle generation takes an increasing sequence of hardness parameters $T_1, \ldots, T_n$ and outputs a sequence of puzzles and solutions $(Z_i, s_i)_{i \in [n]}$. The distinguishing feature is that puzzles can be solved sequentially 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 $T_i - T_{i-1}$. In particular, one does not need to start from the beginning for solving additional puzzles. The interesting aspect now is that the puzzle generator can outsource the solving of the puzzles to an external entity, i.e., a sequential squaring service for TLPs based on sequential squaring. Note that this service has no means to solve the puzzles faster than anyone else (excluding the puzzle generator) and thus there could be a single such service performing the computation instead of requiring to have every receiver doing this wasteful computation on its own.

1.2 Technical Overview and Contributions

Before we discuss our contributions we stress (as already seen in above), that the terms time-lock puzzle (TLP) [31], timed-release encryption (TRE) [27], and time-lock encryption (TLE) [23] are often used interchangeably in the literature to denote an encryption scheme which enables us to send messages in the future. For our constructions we use the former two notions in a slightly different way, and hence we need to distinguish between them. In particular, TRE will denote an encryption scheme which allows us send messages "into the future", while providing confidentiality of message before the release time. A TLP provides the core functionality of a puzzle that needs a certain amount of time to be solved, without considering any messages. TLPs will be used as a building block for TRE. Now, we are ready to discuss our contributions.

Versatile Timed-Release Encryption. We introduce a generic approach to construct TRE. 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 (PKE) scheme. Then, we provide the respective public key pk as parameters of the TRE and solving the puzzle Z reveals s and thus sk allowing to decrypt *all* of the ciphertexts computed with respect to pk. Note that when using s as the random coins for a partially homomorphic encryption scheme, e.g., ElGamal [14], or a fully homomorphic encryption scheme, e.g., BGV [10], this 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. Consequently, solving one puzzle allows to decrypt all ciphertexts associated to a hardness parameter (generated by many potentially independent entities). We note that our approach to HTRE satisfies the basic definition of HTLPs from MT19, where the time required to solve the puzzles starts with the generation of the parameters. In contrast, MT19 also provides the notion of a reusable setup for their LHTLP, where one can use the same parameters to generate many puzzles in a way that for every single puzzle the time only starts to run from the point where the puzzle is generated (this characteristic is also inherited by [9]). However, we observe that for all the applications discussed in [26] 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, MT19 discuss an application to e-voting, where it rather seems to complicate issues when one can only start solving the puzzle after the last voter cast its vote. It seems more practical to set-up the puzzle such that the solution can be made available at a certain pre-defined point in time. 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 (we defer a detailed discussion to the Appendix C).

Moreover, we demonstrate that our TRE framework can be used to obtain other variants of TRE in a generic way. We showcase this using the regime of functional encryption. In particular, we introduce timed-release functional encryption (TRFE) which allows to time-lock a function f. After a certain time has passed everyone can learn the function $f(x)$ of any ever encrypted message x. As an application we discuss identity-based encryption (IBE) [6] 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.

Sustainable Timed-Release Encryption. We introduce the notion of sequential TLPs as a particularly useful generalization of TLPs, which yields practical and particularly sustainable TRE schemes (from the perspective of consumption of computational resources). 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 feature is that puzzles can be solved sequentially 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). From this, we then build a sequential TRE scheme, where security is based on the security of a sequential TLP. Unfortunately, it turns out that such a construction is non-trivial. For a TRE scheme to be secure, we require that any adversary that runs in time $T < T_i$ is not able to break the security of an encryption with respect to time slot T_i. For such an adversary, we need to simulate all values up to T_{i-1}, in particular all TLP 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. However, it is not possible to build a reduction which receives as input $s_1, \ldots, s_{i-1}$ as part of the TLP instance, because then the reduction would only be able to break the assumption that the puzzle is hard if it runs in time less than $T_i - T_{i-1}$ (cf. Section 3 for a detailed discussion). Our solution to overcome this difficulty is to construct a TRE scheme which does not directly use the real solutions s_i, but instead $\mathsf{F}(T_i, s_i)$, where one can think of F as a hard-to-invert function. This way we are able to formulate a hardness assumption for TLPs where the reduction in the security proof of the TRE scheme receives $\mathsf{F}(T_1, s_1), \ldots, \mathsf{F}(T_{i-1}, s_{i-1}), \mathsf{F}(T_{i+1}, s_{i+1}), \ldots, \mathsf{F}(T_n, s_n)$ as additional "advice", and thus is able to provide a proper simulation. At the same time it is reasonable to assume that no adversary is able to distinguish $\mathsf{F}(T_i, s_i)$ from random, even if it runs in time up to $T < T_i$, which is exactly the upper bound that we have on the TRE adversary. We note that MT19 [26, 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 sequential TRE. The technical difficulty that we encounter should arise in their construction as well. Unfortunately, they do not provide a formal security analysis, so that this is not clarified. We, however, believe that a similar assumption involving an "advice" for the reduction is also necessary for a security proof of the construction suggested in their work.

In order to construct sequential TLPs, we introduce the so called *gap sequential squaring assumption*, which extends the sequential squaring assumption by an oracle which takes as input the hardness parameter T' and a value y' and outputs 1 if and only if $y' = x^{2^{T'}} \bmod N$. This is akin to other gap problems [28] such as the well known gap Diffie-Hellman problem. As evidence for the hardness of this assumption, we provide an analysis in the strong algebraic group model (SAGM) and in particular show that our assumption holds as long as factoring is hard. The SAGM was introduced by Katz *et al.* [20] as a variant of the algebraic group model (AGM) [16] and enables to work with time-sensitive assumption. Finally, when modeling the above mentioned function F as a random oracle, we obtain a provably secure construction of a sequential TLP and finally a sequential TRE.

1.3 Summary and Discussion of Properties of Our Approach

There exist different approaches to construct TRE in the literature, which all aim at achieving a similar goal from a high-level perspective, but which provide very different properties from a low-level perspective, with sometimes subtle but crucial differences. Therefore let us briefly summarize the properties achieved by our (sequential) TRE approach again and discuss how it enables novel applications.

Homomorphic TRE. We recall that the *homomorphic* timed-release encryption from MT19 (called HTLPs in [26]) supports homomorphic evaluation of functions on encrypted messages and avoids expensive parallel computations to solve one puzzle per ciphertext, while it still achieves the desired security against time-bounded adversaries. 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 $m_1, \dots, m_n$, but only of their homomorphic evaluation. Also a sufficiently expressive 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 [17] are currently still much less practical. So for applications that require a complex function f, the homomorphic approach from [26] is conceptually interesting, but not yet practical.

Our modular TRE costruction follows a different approach, which 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. It thus can be based on arbitrary hardness assumptions and without introducing further requirements, such as the need for indistinguishability obfuscation for the fully-homomorphic TLP construction in [26], or the need for a specific *multi-key* fully-homomorphic encryption scheme [24] as in [9]. We note also that our construction of sequential TRE equally provides homomorphic computations *within* a single time period. Homomorphic computations *across* different time slots can easily be realized using any *multi-key* homomorphic encryption scheme [24].

Optimal amortized costs. Note that while MT19 [26] achieve the "solve one, get many" property only for homomorphic evaluations over many time-locked messages and thus only for functions evaluated over the time-locked messages, our TRE construction achieves this even without requiring an underlying homomorphic encryption scheme, but for all the original ciphertexts. This is because solving a puzzle yields the randomness to generate the secret key sk of the PKE scheme, which makes it possible to decrypt all ciphertexts efficiently without requiring to solve many puzzles in parallel. Note that the approach of MT19 is thus limited with respect to applicability. The homomorphic evaluation of ciphertexts is only useful when an application needs to decrypt only a function $f(m_1, \dots, m_n)$ of all encrypted messages $m_1, \dots, m_n$. However, if it needs to learn all n messages $m_1, \dots, m_n$ explicitly, then MT19 still requires to solve n puzzles in parallel.

With our TRE approach it is possible to achieve the "solve one, get many" property even for applications that require the full decryption of all independently encrypted encrypted messages. Note that for a number n of independently time-locked messages $m_1, \dots, m_n$ our scheme is thus the first one to achieve an optimal amortized cost of decryption per ciphertext of

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

where $T_{\mathsf{PKE.Dec}}$ is the time required to run the decryption algorithm $\mathtt{PKE}.\mathsf{Dec}$ and T_{TLP} is the time required to solve the puzzle. Note that this approaches $T_{\mathsf{PKE.Dec}}$ with increasing n. We note that this equally applies to our sequential TRE approach.

Public verifiability. In [15], Ephraim *et al.* recently introduced the notion of *public verifiability* for TLPs, meaning that after a party solves the puzzle, they can publish the underlying solution together with a proof which can be later used by anyone to quickly verify the correctness of the solution. Ephraim et al. require this property to hold even if the puzzle is maliciously generated and might have no valid solution. We briefly discuss how our TRE construction provides a public verifiability property, but since in our TRE the puzzles are honestly generated and part of the public parameters, we do not consider malicious puzzle generation. Note that in our TRE construction from the generated puzzle $(Z, s) \leftarrow \mathtt{TLP}.\mathsf{Gen}(T)$ the solution s is used as the random coins to obtain $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s)$. Now, our public TRE parameters include $\mathsf{pp}_e := \mathsf{pk}$ and $\mathsf{pp}_d = Z$ and given a potential solution s' one wants to guarantee that $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s')$ generates the same public key and an equivalent secret key. Therefore, if the used PKE scheme $\mathtt{PKE} = (\mathtt{PKE}.\mathsf{Gen}, \mathtt{PKE}.\mathsf{Enc}, \mathtt{PKE}.\mathsf{Dec})$ provides perfect correctness, this public verifiability property is perfectly satisfied, i.e,. for one pk there cannot be different secret keys output by $\mathtt{PKE}.\mathsf{Gen}$ that behave differently in their decryption behavior. In particular, the publicly verifiable proof is then simply the solution s' and the verification is to check whether $s' \in \mathcal{R}$, to run $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s')$ and to check whether $\mathsf{pk}' = \mathsf{pp}_e$, which represents an efficient check.

Sequential TRE with public servers. One particularly interesting feature of our notion of sequential TRE 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 TRE 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.

We stress that this must not be confused with TRE schemes in a trusted-agent based setting. Loosely speaking, in such schemes a so called time server publishes a single time-dependent trapdoor that then allows decryption of ciphertexts. As shown in [11], this concept is essentially equivalent to identity-based encryption (IBE) [6]. Most importantly, in any such agent-based TRE scheme there is a trusted party which is not only involved in running the setup, but this party then also needs to be online and, in particular, needs to be trusted to keep the secret keys that are supposed to be released at a later point in time confidential until the time has passed. In our approach of sequential TRE with public servers, however, only the setup needs to be trusted. Even for the service that actually performs the squaring there are no shortcuts to revealing the decryption keys before the respective time has passed.

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 $N > 1$ 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., [32]). 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.

1.4 Concurrent and Independent Work

Recently, there have been some independent and concurrent works investigating different aspects of TLPs, which we want to briefly discuss. Most closely related to our work is the one of Katz *et al.* [20] who show that sequential squaring is as hard as factoring in the strong algebraic group model (SAGM) and construct non-malleable timed commitments based upon a novel building block called timed public-key encryption (TPKE). The similarities are that we will also rely on the SAGM to prove the generic hardness of our new gap sequential squaring assumption. However, their TRPKE approach is different to our TRE approach. Firstly, they support a fast and a slow decryption, where former uses the secret key and latter requires solving a TLP. Secondly, while in our setting encryption is efficient, in their TPKE which is constructed from sequential squaring and the Naor-Yung double encryption paradigm one has to compute twice a T-times sequential squaring. This construction achieves CCA security, but they also discuss a CPA secure version where encryption is equivalently expensive. Note that in contrast to our TRE, in their TPKE time starts running with encryption and not with parameter generation.

In [15] Ephraim *et al.* investigate efficient constructions of concurrent non-malleable TLPs in the auxiliary-input random oracle model (whereas previous constructions in the plain model [4] are not practically efficient). The idea, which is similar to our idea, is essentially to evaluate random oracle on hardness parameter T and solution s of a puzzle Z and use the output of the oracle as a randomness for Gen algorithm of any TLP. An interesting property introduced and investigated in [15] is public verifiability of TLPs. As we have already discussed we can achieve public verifiability for our generic TRE when basing them on perfectly correct PKE schemes. We note however that while Ephraim et al. consider this notion in a setting with malicious puzzle generation, we consider a weaker notion with honest puzzle generation which is sufficient for our TRE.

Abadi and Kiayias [1] construct so-called Multi-Instance Time-Lock Puzzles (MITLPs) which are similar to our notion of sequential timed-release encryption. The crucial difference is that MILTPs allow to encrypt messages with respect to consecutive multiples of one hardness parameter by chaining TLPs which requires that all messages of interest must be known at the time when MITLP is generated.

1.5 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 sequential time-lock puzzles and provide instantiations of sequential time-lock puzzles from a variant of the sequential squaring problem in Section 3. Finally, Section 4 presents our generic construction of timed-release encryption, also covering sequential homomorphic and functional variants thereof. As already mentioned, we defer the discussion of applications to the Appendix C.

2 Definitions and Constructions of Time Lock-Puzzles

Before we start with the definitions, we introduce some required notation. We denote our security parameter as λ. 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 $x \overset{\$}{\leftarrow} S$ to indicate that we choose x uniformly at 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 λ, $a_1, \ldots, a_n$ and fresh randomness and then assigning the output to x (we will usually omit λ and assume that all algorithms take λ 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.

2.1 Simple Time-Lock Puzzles

In this section we give a new definition for time-lock puzzles (TLPs) 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* Gen *runs in time at most* $\mathsf{poly}(\log T, \lambda)$ *for some polynomial* 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* 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* λ *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 TLPs from Bitansky *et al.* [5] and Malavolta and Thyagarajan [26] algorithm Gen receives s as an additional input and output a puzzle Z. This immediately yields a timed-release encryption (TRE) scheme by viewing s as a message that is encrypted. Our definition enables a slightly simpler generic construction of (homomorphic) TRE. 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 $\texttt{TLP} = (\mathsf{Gen}, \mathsf{Solve})$ be a puzzle according to our new definition. Then we obtain a puzzle $\texttt{TLP}' = (\mathsf{Gen}', \mathsf{Solve}')$ 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 TLP 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.* [5] and Malavolta and Thyagarajan [26], but adopted to our slightly modified definition of the Gen algorithm.

Definition 2. *Consider the security experiment* $\mathsf{ExpTLP}^b_{\mathcal{A}}(\lambda)$ *in Figure 1. We say that a time-lock puzzle* $\texttt{TLP}$ *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*

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

Other Variants of TLPs. We briefly discuss weaker forms of TLPs as introduced by Bitansky *et al.* [5]. First, weak TLPs (wTLPs) that do not require that Gen can be computed in time $\mathsf{poly}(\log T, \lambda)$, but either in fast parallel time (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 Gen then runs in (sequential) time $\mathsf{poly}(\log T, \lambda)$.

Definition 3 (Weak Time-Lock Puzzles [5]). *A weak time-lock puzzle (wTLP)* $\texttt{wTLP} = (\mathsf{Gen}, \mathsf{Solve})$ *is satisfying the syntax and completeness requirements as per Definition 1, but with the following efficiency requirements:* Gen *can be computed by a uniform circuit of size* $\mathsf{poly}(T, \lambda)$ *and depth* $\mathsf{poly}(\log T, \lambda)$ *and* Solve *can be computed in time* $T \cdot \mathsf{poly}(\lambda)$.

Definition 4 (Time-Lock Puzzles with Pre-processing [5]). *A time-lock puzzle with pre-processing (ppTLP) is a tuple of algorithms* $\mathsf{ppTLP} = (\mathsf{Preproc}, \mathsf{Gen}, \mathsf{Solve})$*:*

- $(P, K) \leftarrow \mathsf{Preproc}(T)$ *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* $\mathsf{poly}(\log T, \lambda)$.

```
$\mathsf{ExpTLP}^b_{\mathcal{A}}(\lambda)$:
$(Z, s) \leftarrow \mathsf{Gen}(T(\lambda)), b \xleftarrow{\$} \{0,1\}$
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.

– $s \leftarrow \mathsf{Solve}(P, Z)$ *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* λ*, for all polynomials* T *in* λ *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.

2.2 Instantiating TLPs from Sequential Squaring

Subsequently, we discuss instantiations of TLPs based on the sequential squaring. Therefore, we recall a definition of the sequential squaring assumption which was implicitly introduced by Rivest et al. [31]. 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 GenMod be a probabilistic polynomial-time algorithm which, on input 1^λ, outputs two λ-bit strong primes p and q and modulus N that is the product of p and q. Let $\varphi(\cdot)$ denotes Euler's totient function. We denote by $\mathbb{QR}_N$ the cyclic group of quadratic residues which has order $|\mathbb{QR}_N| = \frac{\varphi(N)}{4} = \frac{(p-1)(q-1)}{4}$.

Definition 5 (The Sequential Squaring Assumption). *The* sequential squaring assumption *with gap* $0 < \epsilon < 1$ *holds relative to* GenMod *if there exists a polynomial* $\tilde{T}(\cdot)$ *such that for all polynomials* $T(\cdot) \geq \tilde{T}(\cdot)$ *and for every non-uniform polynomial-size adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$*, where the depth of* $\mathcal{A}_\lambda$ *is at most* $T^\epsilon(\lambda)$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$

$$\left| \Pr\left[b = b' \ : \ \begin{array}{r} (p, q, N) \leftarrow \mathsf{GenMod}(1^\lambda) \\ x \xleftarrow{\$} \mathbb{QR}_N, b \xleftarrow{\$} \{0,1\} \\ \textit{if } b = 0 : y := x^{2^{T(\lambda)}} \bmod N \\ \textit{if } b = 1 : y \xleftarrow{\$} \mathbb{QR}_N \\ b' \leftarrow \mathcal{A}_\lambda(N, T(\lambda), x, y) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

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

- $\mathsf{Gen}(T)$: Run $(p, q, N) \leftarrow \mathsf{GenMod}(1^\lambda)$. Randomly sample $x \overset{\$}{\leftarrow} \mathbb{QR}_N$ and compute the value $s := x^{2^T} \bmod N$. 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 $s := x^{2^T} \bmod N$ by repeated squaring.

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

2.3 Instantiating TLPs from Randomized Encodings

Subsequently, we discuss instantiations of TLPs based on different variants of randomized encodings [19,3] and in particular the approach of constructing TLPs from them by Bitansky et al. [5].

We first recall TLPs from randomized encodings (REs) in [5] 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. [21]).

First, we recall a TLP $\mathtt{TLP}' = (\mathsf{Gen}', \mathsf{Solve}')$ as defined in [5], 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 $s \overset{\$}{\leftarrow} S$ 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 [5] (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 6 (Randomized Encoding [5]). *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 [5]. Using the fact that garbled circuits yield randomized encodings (cf. e.g., for discussion [2]), we have the following:

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

Definition 7 (Reusable Randomized Encoding [5]). *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 [5].

Theorem 1 ([18]). *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 3λ bits (which is possible for large enough λ). 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 [5]). *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* ϵ. *Then, for any* $\epsilon' < \epsilon$*, the above construction is a weak time-lock puzzle with gap* ϵ'.

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 [5]). *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 ϵ. Then, for any $\epsilon' < \epsilon$, the above construction is a time-lock puzzle with pre-processing with gap ϵ'.*

Remark 2. As mentioned in [26], 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 Sequential Time-Lock Puzzles

In this section we introduce *sequential* time-lock puzzles along with their security and propose an instantiation which we prove secure under a new assumption called the *gap sequential squaring* assumption. We also show this assumption to hold, assuming factoring is hard, in the strong algebraic group model (SAGM) of Katz *et al.* [20].

3.1 Defining Sequential Time-Lock Puzzles

Sequential time-lock puzzles are a particularly useful generalization of basic TLPs, which yields particularly practical time-lock encryption schemes. To this end, we generalize Definition 1 by allowing the Gen algorithm to take multiple different time parameters as input, which then produces a corresponding set of puzzles.

Definition 8. *A* sequential time-lock puzzle *is tuple of algorithms* $\mathtt{sTLP} = (\mathsf{Gen}, \mathsf{Solve})$ *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 $s_0 := \bot$. We require that* Solve *runs in time at most* $(T_i - T_{i-1}) \cdot \mathsf{poly}(\lambda)$, *where we define $T_0 := 0$.*

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

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

```
$\mathsf{ExpsTLP}^b_{\mathcal{A}_i}(\lambda)$:
$(Z_j, s_j)_{j\in[n]} \leftarrow \mathsf{Gen}(1^\lambda, (T_j(\lambda))_{j\in[n]})$
$(y_j := \mathsf{F}(T_j(\lambda), s_j))_{j\in\{[n]\setminus\{i\}\}}$
if $b = 0 : y_i := \mathsf{F}(T_i(\lambda), s_i)$
if $b = 1 : y_i \overset{\$}{\leftarrow} Y$
return $b' \leftarrow \mathcal{A}_{i,\lambda}((Z_j, y_j)_{j\in[n]})$
```

Fig. 2. Security experiment for sequential time-lock puzzles.

Security. In order to define a security notion for sequential time-lock puzzles that is useful for our application of constructing particularly efficient timed-release encryption schemes, we need to introduce an additional function $\mathsf{F} : \mathbb{N} \times S \to Y$, which takes as input a pair a hardness parameter $T \in \mathbb{N}$ together with solution $s \in S$ and outputs elements of some set Y. Instead of requiring that elements s_i are indistinguishable from random, we require that $y_i = \mathsf{F}(T_i, s_i)$ is indistinguishable from random. We explain the necessity for function F after the following definition.

Definition 9. *Consider the security experiment* $\mathsf{ExpsTLP}^b_{\mathcal{A}_i}(\lambda)$ *in Figure 2. We say that a sequential time-lock puzzle* `sTLP` *is* secure with gap $0 < \epsilon < 1$ and with respect to the function F, *if for all polynomials n in λ there exists a polynomial $\tilde{T}(\cdot)$ such that for all sets of polynomials $(T_j(\cdot))_{j\in[n]}$ fulfilling that $\forall j \in [n] : T_j(\cdot) \geq \tilde{T}(\cdot)$, for all $i \in [n]$ and every polynomial-size adversary $\mathcal{A}_i = \{\mathcal{A}_{i,\lambda}\}_{\lambda\in\mathbb{N}}$, where the depth of $\mathcal{A}_{i,\lambda}$ 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*

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

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

In Section 4.2 we will build a sequential timed-release *encryption* (TRE) scheme, where security is based on the security of a sequential TLP. We will consider this scheme insecure, if an adversary that runs in time $T < T_i$ is able to break the security of an encryption with respect to time slot T_i (see Definition 14). For such an adversary, we need to simulate all values up to T_{i-1} and in particular all TLP solutions $s_1, \ldots, s_{i-1}$ up to T_{i-1}, properly. 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 TLP 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 $T_i > T \geq T_i - T_{i-1}$.

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 TRE scheme which does not directly use the real solutions s_i, but instead $\mathsf{F}(T_i, s_i)$ where one can think of F as a hard-to-invert function. This way we are able to formulate a hardness assumption for TLPs where the reduction in the security proof of the TRE scheme receives $\mathsf{F}(T_1, s_1), \ldots, \mathsf{F}(T_{i-1}, s_{i-1}), \mathsf{F}(T_{i+1}, s_{i+1}), \ldots, \mathsf{F}(T_n, s_n)$ as additional "advice" that can be used to provide a proper simulation. At the same time it is reasonable to assume that no adversary (our reduction, here) is able to distinguish $\mathsf{F}(T_i, s_i)$ from random, even if it runs in time up to $T < T_i$, which is exactly the upper bound that we have on the TRE adversary.

We again note that the work of Malavolta and Thyagarajan [26, Section 5.2] also proposes an approach that allows to use multiple time slots, by describing a specific construction which is similar to our notion of sequential TRE. 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 the construction suggested in their work.

3.2 Instantiating Sequential TLPs from Sequential Squaring

In order to obtain a sequential TLP, we define a variant of the sequential squaring assumption in which an adversary is given oracle access to a *decisional sequential squaring verification function* DSSvf. DSSvf takes as input hardness parameter T' and value $y' \in \mathbb{QR}_N$ and outputs 1 if $y' = x^{2^{T'}} \bmod N$, otherwise it outputs 0. The values x and N are defined in security experiment. The assumption essentially states that computational sequential squaring assumption remains hard, even if the adversary is given access to DSSvf, akin to other gap assumptions [28]. We discuss the necessity of this assumption in Appendix A.

Definition 10 (The Gap Sequential Squaring (GGS) Assumption). *The* gap sequential squaring assumption *with gap $0 < \epsilon < 1$ holds relative to* GenMod *if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and for every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, where the depth of $\mathcal{A}_\lambda$ is 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*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathsf{GSS}} = \Pr\left[y = x^{2^T} \bmod N \colon \begin{array}{r} (p, q, N) \leftarrow \mathsf{GenMod}(1^\lambda), x \xleftarrow{\$} \mathbb{QR}_N \\ y \leftarrow \mathcal{A}_\lambda^{\mathsf{DSSvf}(\cdot,\cdot)}(N, T(\lambda), x) \end{array} \right] \leq \mathsf{negl}(\lambda),$$

where $\mathsf{DSSvf}(\cdot,\cdot)$ *is an oracle which takes as input a hardness parameter T' and a value y' and outputs 1 if and only if $y' = x^{2^{T'}} \bmod N$.*

Now we are ready to construct our sequential TLP:

– $\mathsf{Gen}((T_i)_{i \in [n]})$: Run $(p, q, N) \leftarrow \mathsf{GenMod}(1^\lambda)$. Randomly sample $x \xleftarrow{\$} \mathbb{QR}_N$ and compute values $s_i := x^{2^{T_i}} \bmod N$ for all $i \in [n]$. Value s_i can be efficiently computed knowing the values p and q. Output $((N, x, T_i, T_{i-1}), s_i))_{i \in [n]}$.

– $\mathsf{Solve}((N, x, T_i, T_{i-1}), s_{i-1})$: Compute value $s_{i-1}^{T_i - T_{i-1}} \bmod N$ by repeated squaring.

Theorem 4. *If the gap sequential squaring assumption with gap ϵ holds relative to* GenMod *and* F *is modelled as a random oracle, then for any $\underline{\epsilon} < \epsilon$, the* $\mathtt{sTLP} = (\mathsf{Gen}, \mathsf{Solve})$ *defined above is a secure sequential time-lock puzzle with gap $\underline{\epsilon}$ and with respect to the function* F.

Proof. Let $\tilde{T}_{\mathtt{GSS}}(\lambda)$ be the polynomial whose existence is guaranteed by the GSS assumption. Let $\mathsf{poly}_{\mathsf{RO}}(\lambda)$ be the fixed polynomial which bounds the time required to execute Step 1 and simulate random oracle answers as specified in Step 2 of the adversary $\mathcal{B}_\lambda$ defined below. Set $\underline{T}(\lambda) := (\mathsf{poly}_{\mathsf{RO}}(\lambda))^{1/\underline{\epsilon}}$. Set $\tilde{T}_{\mathtt{sTLP}} := \max(\tilde{T}_{\mathtt{GSS}}, \underline{T})$

For any n which is polynomial in λ, any tuple $(T_j(\cdot))_{j\in[n]}$ fulfilling that $\forall j \in [n]$ holds $T_j(\cdot) \geq \tilde{T}_{\mathtt{sTLP}}(\cdot)$, any $i \in [n]$, from any polynomial-size adversary $\mathcal{A}_i = \{\mathcal{A}_{i,\lambda}\}_{\lambda\in\mathbb{N}}$, where the depth of $\mathcal{A}_{i,\lambda}$ is bounded from above by $T_i^{\underline{\epsilon}}(\lambda)$, we can construct a polynomial-size adversary $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda\in\mathbb{N}}$ whose depth is bounded from above by $T_i^{\epsilon}(\lambda)$ such that

$$\mathbf{Adv}_{\mathcal{B}}^{\mathtt{GSS}} = \mathbf{Adv}_{\mathcal{A}_i}^{\mathtt{sTLP}}.$$

Intuitively, an adversary cannot distinguish $\mathsf{F}(T_i, s_i)$ from random without asking (T_i, s_i) to F, since F is a random oracle. More formally, let Query be the event that $\mathcal{A}_i$ asks (T_i, s_i) to F, where s_i is the correct solution of the puzzle (N, T_i, x). Then, by arguing as in Shoup's Difference Lemma [34], we get:

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}_i}^{\mathtt{sTLP}} &= \left|\Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1\right] - \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1\right]\right| \\
&= \Big| \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \mathsf{Query}\right] + \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \overline{\mathsf{Query}}\right] \\
&\qquad - \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \mathsf{Query}\right] - \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \overline{\mathsf{Query}}\right] \Big| \\
&= \Big| \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \mathsf{Query}\right] - \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \mathsf{Query}\right] \Big| \\
&\leq \Pr[\mathsf{Query}].
\end{aligned}$$

The second equation follows from the fact that if Query does not happen then the value y_i is uniformly distributed in both experiments. Therefore the event $\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \overline{\mathsf{Query}}$ occurs if and only if $\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \overline{\mathsf{Query}}$, so that we get

$$\Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \overline{\mathsf{Query}}\right] = \Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \overline{\mathsf{Query}}\right].$$

The last inequality follows from the fact that both $\Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{0}(\lambda) = 1 \wedge \mathsf{Query}\right]$ and $\Pr\left[\mathsf{ExpsTLP}_{\mathcal{A}_i}^{1}(\lambda) = 1 \wedge \mathsf{Query}\right]$ are numbers between 0 and $\Pr[\mathsf{Query}]$.

Hence, in order to complete the proof, we show that $\Pr[\mathsf{Query}]$ is negligible, provided that the gap sequential squaring (GSS) assumption holds. To this end, we define the following adversary $\mathcal{B}$ on the GSS problem:

1. $\mathcal{B}$ receives (N, T, x) as input. It sets $T_i := T$ and $Z_j := (N, T_j, T_{j-1}, x)_{j\in[n]}$ and for all $j \in [n]$ randomly samples values y_j from the image of F.

2. $\mathcal{B}$ runs $\mathcal{A}_i$ on input $((Z_j, y_j)_{j\in[n]})$. It initializes an empty list Q. When $\mathcal{A}_i$ makes a query (T', s') to F, answer it as follows:
   - If there is an entry in Q of the form $((T', s'), y)$ for some y, return y.
   - Else if $T' = T_j$ for some $j \in [n]$ and $\mathsf{DSSvf}(T', s') = 1$, store $((T', s'), y_j)$ in Q. If $i = j$ then output s' as a solution to the GSS problem. Otherwise, return y_j to $\mathcal{A}_i$.
   - Otherwise, sample a uniform value y from the image of F, store $((T', s'), y)$ in Q and return y to $\mathcal{A}_i$.

Notice that $\mathcal{B}$ consistently simulates the random oracle for $\mathcal{A}_i$. Moreover, when Query occurs, then $\mathcal{B}$ outputs a correct solution to the GSS problem. Hence,

$$\Pr[\mathsf{Query}] \leq \mathbf{Adv}^{\mathtt{GSS}}_{\mathcal{B}}.$$

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

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

On the other hand $T(\cdot) \geq \tilde{T}_{\mathtt{sTLP}}(\cdot) \geq \tilde{T}_{\mathtt{GSS}}(\cdot)$ as required.
This concludes the proof. $\square$

3.3 Hardness of the Gap Sequential Squaring Assumption

In this section we show that the gap sequential squaring problem is at least as hard as factoring N in the Strong Algebraic Group Model (SAGM). The SAGM was introduced by Katz *et al.* [20] as a variant of the Algebraic Group Model (AGM) [16] and makes it possible to consider time-sensitive assumptions, such as (Gap) Sequential Squaring. In the SAGM the running time of an algebraic algorithm is defined by the number of its group operations with respect to some cyclic group G. In our case G is the group $\mathbb{QR}_N$ of quadratic residues modulo N, for some N output by GenMod. Therefore, we use multiplication to denote the group operation. In the strong AGM, all algorithms are treated as strongly algebraic.

Definition 11 (Strongly algebraic algorithm [20]). *An algorithm $\mathcal{A}$ over $\mathbb{G}$ is called* strongly algebraic*, if in each (algebraic) step $\mathcal{A}$ does arbitrary local computation and then outputs*[3] *one or more tuples of the following form:*

1. $(y, y_1, y_2) \in \mathbb{G}^3$*, where $y = y_1 y_2$ and y_1, y_2 were either previously received by $\mathcal{A}$ or output by $\mathcal{A}$ in previous steps;*
2. $(y, y_1) \in \mathbb{G}^2$*, where $y = y_1^{-1}$ and y_1 was either provided as input to $\mathcal{A}$ or was output by $\mathcal{A}$ in some previous step.*

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

[3] Here it is required that $\mathcal{A}$ outputs group elements at intermediate steps of its computation. The final output of $\mathcal{A}$ can be distinguished by requiring $\mathcal{A}$ to output a special indicator when generating its final output.

Parallel computation is captured in the SAGM by allowing an algorithm to output multiple group elements at once (in one step, which counts as one query), but it is required that all of them can be represented by a sequence of previously output group elements. There are two ways how the running time of an adversary is measured: the number of group operations and the standard running time in some underlying computational model. Hence the running time is defined as a pair (t_1, t_2), which is understood in the following sense: running in time t_2 and executing t_1 group operations. For a more detailed description of SAGM and its relation to AGM and the Generic Group Model (GGM) [33] we refer to the original paper by Katz *et al.* [20].

Definition 12 (The Factoring Assumption). *The Factoring Assumption holds relative to* GenMod *if for every polynomial-size adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *it holds*

$$\mathbf{Adv}^{\mathtt{Factor}}_{\mathcal{A}} = \Pr\left[N = p'q' \ : \ \begin{array}{r} (p,q,N) \leftarrow \mathsf{GenMod}(1^\lambda) \\ p', q' \leftarrow \mathcal{A}(N), \\ \textit{such that } p', q' \in \mathbb{N}; p', q' > 1 \end{array} \right] \leq \mathsf{negl}(\lambda).$$

In order to prove the hardness of the Gap Sequential Squaring Problem, we use the following well-known fact, which states that we can factor N if a multiple of $\varphi(N)$ is known. The following lemma extends Lemma 1 of [20] from a success probability of $1/2$ to a success probability negligibly close to 1, which will be required for our proof.

Lemma 1. *Let* $(p,q,N) \leftarrow \mathsf{GenMod}(1^\lambda)$ *and let* $m = \alpha\varphi(N)$ *for some positive integer* $\alpha \in \mathbb{Z}^+$*. There exists a PPT algorithm* $\mathsf{Factor}(N,m)$ *which, on input* (N,m)*, outputs* $p', q' \in \mathbb{N}$*,* $p', q' > 1$ *such that* $N = p'q'$ *with probability at least* $1 - 2^{-\lambda}$.

Proof. Lemma 1 of [20] guarantees an existence of efficient $\mathsf{Factor}(N,m)$ algorithm which has success probability at least $1/2$. By running this algorithm λ-times with independent randomness, we obtain claimed success probability. □

Now we are ready to show that the Gap Sequential Squaring Problem is hard relative to GenMod for any T and for adversaries which perform at most $T-1$ group operations.

Theorem 5. *If the factoring assumption holds relative to* GenMod*, then the gap sequential squaring assumption with gap* ϵ *holds relative to* GenMod *in the SAGM for any* $0 < \epsilon < 1$.

Proof. We show that for any positive integer T the gap sequential squaring assumption holds against adversaries which perform at most $T-1$ algebraic steps. Notice that this means that the gap $\epsilon = \log_T(T-1)$. Since $\lim_{T\to\infty} \log_T(T-1) = 1$, ϵ can be defined arbitrary close to 1 by choosing $\widetilde{T}$ appropriately. Let $\mathcal{A}$ be a strongly algebraic adversary which executes at most $T-1$ algebraic steps and asks at most ℓ queries to DSSvf oracle. Let $x \in \mathbb{QR}_N$ be the group element which is given to $\mathcal{A}$ as part of its challenge. Following [20], we recursively define for any $y \in \mathbb{QR}_N$ output by $\mathcal{A}$ the discrete logarithm of y with respect to $\mathcal{A}$ and x, $\mathsf{DL}_{\mathcal{A}}(x,y) \in \mathbb{Z}^+$ as follows:

1. $\mathsf{DL}_{\mathcal{A}}(x,x) = 1$,

2. If $\mathcal{A}$ outputs (y, y_1, y_2) in a certain step, then $\mathsf{DL}_{\mathcal{A}}(x, y) = \mathsf{DL}_{\mathcal{A}}(x, y_1) + \mathsf{DL}_{\mathcal{A}}(x, y_2)$.
3. If $\mathcal{A}$ outputs (y, y_1) in an algebraic step, then $\mathsf{DL}_{\mathcal{A}}(x, y) = -\mathsf{DL}_{\mathcal{A}}(x, y_1)$.

Now we can make the following observation. Assume that x is a generator of $\mathbb{QR}_N$. If $\mathcal{A}$ outputs a solution s' of a puzzle (N, T', x) in one of its queries to the DSSvf oracle or as solution to the puzzle challenge, and at the same time it holds that $\mathsf{DL}_{\mathcal{A}}(x, s') \neq 2^{T'}$, then $|\mathbb{QR}_N|$ divides $2^{T'} - \mathsf{DL}_{\mathcal{A}}(x, s')$. This is because $1 = s'(s')^{-1} = x^{2^{T'}}(x^{\mathsf{DL}_{\mathcal{A}}(x,s')})^{-1}$ $= x^{2^{T'} - \mathsf{DL}_{\mathcal{A}}(x,s')} \bmod N$. Because $\mathsf{DL}_{\mathcal{A}}(x, s') \neq 2^{T'}$ this implies that $2^{T'} - \mathsf{DL}_{\mathcal{A}}(x, s')$ is a multiple of the order of the group $\mathbb{QR}_N$. Recall that $|\mathbb{QR}_N| = \varphi(N)/4$. Hence $4(2^{T'} - \mathsf{DL}_{\mathcal{A}}(x, s'))$ is multiple of $\varphi(N)$. By Lemma 1 we are able to factor N with probability at least $1 - 2^{-\lambda}$.

With this in mind, we are ready to construct an adversary $\mathcal{B}$ breaking the Factoring Assumption:

1. $\mathcal{B}$ receives N as input. It samples randomly $x \xleftarrow{\$} \mathbb{QR}_N$ and runs $\mathcal{A}$ on input (N, T, x).
2. Whenever $\mathcal{A}$ asks a query (T', s') to DSSvf, $\mathcal{B}$ recursively computes $\mathsf{DL}_{\mathcal{A}}(x, s')$ and proceeds in the following way:
   (a) If $2^{T'} = \mathsf{DL}_{\mathcal{A}}(x, s')$ then it returns 1 to $\mathcal{A}$.
   (b) Otherwise, it sets $m := 4(2^{T'} - \mathsf{DL}_{\mathcal{A}}(x, s'))$ and executes the factoring algorithm from Lemma 1. If the factoring algorithm is successful, it outputs the corresponding factors as a solution to the factoring problem. Else it outputs 0 to $\mathcal{A}$.
3. When $\mathcal{A}$ returns a solution s, $\mathcal{B}$ computes $\mathsf{DL}_{\mathcal{A}}(x, s)$, sets $m := 4(2^{T'} - \mathsf{DL}_{\mathcal{A}}(x, s'))$, calls the factoring algorithm from Lemma 1 and returns whatever this algorithm returns.

Let us analyse the success probability of $\mathcal{B}$. Let FACTOR be the event that $\mathcal{B}$ successfully outputs the factorization of N and GSS be the event that $\mathcal{A}$ outputs the correct solution of the Gap Sequential Squaring Problem. Let GNR denote the event that the sampled x in Step 1 is a generator. Because x is sampled uniformly at random and $\mathbb{QR}_N$ has $\varphi(|\mathbb{QR}_N|) = (p' - 1)(q' - 1)$ generators, this event happens with overwhelming probability. Concretely, $\Pr[\mathsf{GNR}] = 1 - \frac{1}{p'} - \frac{1}{q'} + \frac{1}{p'q'}$.

In Step 2(b) we have that $2^{T'} \neq \mathsf{DL}_{\mathcal{A}}(x, s')$, and hence if s' was a solution of the given puzzle, then we should be able to factor N with probability at least $1 - 2^{-\lambda}$ by Lemma 1. Notice that $\mathcal{B}$ answers the DSSvf-query incorrectly only if s' is a solution of the puzzle and the factoring algorithm was unsuccessful. Let FAIL_i denotes the event that $\mathcal{B}$ answered the i-th DSSvf query incorrectly and let FAIL denote the event that it answered any of the l queries incorrectly. Then $\Pr[\mathsf{FAIL}_i | \mathsf{GNR}] \leq \frac{1}{2^\lambda}$. Hence, the probability of FAIL can be upper bounded by a union bound:

$$\Pr[\mathsf{FAIL} | \mathsf{GNR}] = \Pr[\bigvee_{i=1}^{l} \mathsf{FAIL}_i | \mathsf{GNR}] \leq \frac{l}{2^\lambda}.$$

We obtain that $\mathcal{B}$ answers all DSSvf-queries correctly with probability

$$\Pr[\overline{\mathsf{FAIL}} | \mathsf{GNR}] = 1 - \Pr[\mathsf{FAIL} | \mathsf{GNR}] \geq 1 - \frac{l}{2^\lambda}.$$

If the event FAIL does not occur, then $\mathcal{B}$ perfectly simulates the Gap Sequential Squaring experiment. Perfect simulation guarantees that in Step 3, $\mathcal{A}$ outputs a correct solution with probability $\mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}}$. Therefore $\Pr[\mathsf{GSS}|\overline{\mathsf{FAIL}}] = \mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}}$. However, we are interested in $\Pr[\mathsf{GSS}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]$, which can be bounded as follows:

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}} &= \Pr[\mathsf{GSS}|\overline{\mathsf{FAIL}}] \\ &= \Pr[\mathsf{GSS}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]\Pr[\mathsf{GNR}] + \Pr[\mathsf{GSS}|\overline{\mathsf{GNR}} \wedge \overline{\mathsf{FAIL}}]\Pr[\overline{\mathsf{GNR}}] \\ &\leq \Pr[\mathsf{GSS}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]\Pr[\mathsf{GNR}] + \Pr[\overline{\mathsf{GNR}}].\end{aligned}$$

Hence,

$$\Pr[\mathsf{GSS}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]\Pr[\mathsf{GNR}] \geq \mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}} - \Pr[\overline{\mathsf{GNR}}] = \mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}} - \frac{1}{p'} - \frac{1}{q'} + \frac{1}{p'q'}.$$

Now, because $\mathcal{A}$ can perform at most $T-1$ group operations and the only group element given as input is x, it must hold $\mathsf{DL}_{\mathcal{A}}(x,s) < 2^{T-1}$ (this is formally proven in [20, Lemma 2]). Therefore $2^T \neq \mathsf{DL}_{\mathcal{A}}(x,s)$, which implies that $\varphi(N)|4(2^T - \mathsf{DL}_{\mathcal{A}}(x,s))$. Therefore, by using Lemma 1 we are able to factor N with probability at least $1-2^{-\lambda}$. Hence, $\Pr[\mathsf{FACTOR}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}} \wedge \mathsf{GSS}] \geq 1-2^{-\lambda}$. We conclude that

$$\begin{aligned}&\mathbf{Adv}_{\mathcal{B}}^{\mathtt{Factor}} = \Pr[\mathsf{FACTOR}] \\ &= \Pr[\mathsf{FACTOR}|\mathsf{GNR}]\Pr[\mathsf{GNR}] + \Pr[\mathsf{FACTOR}|\overline{\mathsf{GNR}}]\Pr[\overline{\mathsf{GNR}}] \\ &\geq \Pr[\mathsf{FACTOR}|\mathsf{GNR}]\Pr[\mathsf{GNR}] \\ &\geq \Pr[\mathsf{FACTOR}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]\Pr[\overline{\mathsf{FAIL}}|\mathsf{GNR}]\Pr[\mathsf{GNR}] \\ &\geq \Pr[\mathsf{FACTOR}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}} \wedge \mathsf{GSS}]\Pr[\mathsf{GSS}|\mathsf{GNR} \wedge \overline{\mathsf{FAIL}}]\Pr[\overline{\mathsf{FAIL}}|\mathsf{GNR}]\Pr[\mathsf{GNR}] \\ &\geq (1-\frac{1}{2^\lambda})(1-\frac{l}{2^\lambda})(\mathbf{Adv}_{\mathcal{A}}^{\mathtt{GSS}} - \frac{1}{p'} - \frac{1}{q'} + \frac{1}{p'q'}),\end{aligned}$$

which completes the proof. □

Extension to the *Strong* Sequential Squaring Assumption. Malavolta *et al.* [26] and Katz *et al.* [20] also consider the so-called *Strong* Sequential Squaring Assumption, which involves a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. Adversary $\mathcal{A}_0$ receives a modulus N as input at the beginning of the game. It may perform an *unbounded* amount of computations and produces a state st. Then $\mathcal{A}_1$ is a *bounded* algorithm, which receives as input st and the sequential squaring instance x. Katz *et al.* [20] show that the *Strong* Sequential Squaring assumption holds in the SAGM, assuming that factoring N is hard.

One could similarly define a Strong *Gap* Sequential Squaring assumption, with a two-stage adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ where $\mathcal{A}_0$ is unbounded, but independent of the sequential squaring challenge, while $\mathcal{A}_1$ is bounded as in [20] and additionally has access to a DSSvf oracle as defined in Definition 10. We remark that our proof of hardness of the GSS problem can also be adopted to this setting with a partially unbounded adversary. We do not require this in our work, but it might be useful for future applications.

4 (Sequential) Timed-Release Encryption

In this section we give generic constructions of (sequential) timed-release encryption (TRE) schemes based on (sequential) TLPs. There exist several definitions for TRE and we base ours on that of Unruh [36]. However, we introduce two additional algorithms Setup and Solve which leads to better modularity and applicability of TRE, as we will illustrate in Appendix C.

Definition 13. *A sequential timed-release encryption scheme with message space $\mathcal{M}$ 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^λ 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* $\mathsf{PP} := (\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_{i\in[n]}$. *We require that* 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* 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 $m \in \mathcal{M}$, and outputs a ciphertext c.*
- $m/\perp \leftarrow \mathsf{Dec}(T_i, s_i, c)$ *is a deterministic algorithm which takes as input a hardness parameter T_i, a solution s_i and a ciphertext c, and outputs $m \in \mathcal{M}$ or $\perp$.*

We say a sequential timed-release encryption scheme is correct *if for all $\lambda, n \in \mathbb{N}$, for all sets of hardness parameters $(T_j)_{j\in[n]}$ such that $\forall j \in [n-1] : T_j < T_{j+1}$, for all $i \in [n]$ and for all messages $m \in \mathcal{M}$ it holds:*

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

Note that the above definition also defines "non-sequential" TRE, by setting $n = 1$. In that case the value T_i is not needed as an input for Dec algorithm, however, for sequential TRE, this value is necessary. For ease of the notation, it is unified.

Definition 14. *A sequential timed-release encryption scheme is* secure with gap $0 < \epsilon < 1$ *if for all polynomials n in λ there exists a polynomial $\tilde{T}(\cdot)$ such that for all sets of polynomials $(T_j)_{j\in[n]}$ fulfilling that $\forall j \in [n] : T_j(\cdot) \geq \tilde{T}(\cdot)$, for all $i \in [n]$ and every polynomial-size adversary $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda})\}_{\lambda\in\mathbb{N}}$ there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$ it holds*

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

We require that $|m_0| = |m_1|$ and an adversary $\mathcal{A}_\lambda = (\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda})$ where $\mathcal{A}_{1,\lambda}$ outputs i in the second step of the above security experiment consists of two circuits with total *depth at most $T_i^\epsilon(\lambda)$ (i. e., the total depth is the sum of the depth of $\mathcal{A}_{1,\lambda}$ and $\mathcal{A}_{2,,\lambda}$).*

4.1 Basic TRE Construction

Building blocks. Our construction combines a time-lock puzzle (TLP) with a CPA secure public-key encryption (PKE) scheme.

Definition 15. *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^λ outputs a public/secret key pair.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ *is a probabilistic algorithm that takes as input a public key* 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* 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:*

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

Definition 16. *A* PKE *scheme is CPA secure if for all non-uniform PPT adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ there is a negligible function* negl *such that*

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

where we require that $|m_0| = |m_1|$.

We require standard CPA security of the PKE scheme, since this is sufficient to construct a TRE scheme achieving Definition 14. A stronger CCA-style security notion for TRE would be achievable by replacing the below definition with CCA security. However, we consider this as not very useful for TRE, 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 a CCA-secure public-key encryption scheme, for instance [4].

[4] By replacing the PKE with a CCA secure one, we can straightforwardly obtain a CCA secure TRE. This is easily achieved as our puzzles are included in the public parameters and thus do not need to be non-malleable and we only require non-malleability on the ciphertexts.

Construction. Let $\mathtt{TLP} = (\mathtt{TLP}.\mathsf{Gen}, \mathtt{TLP}.\mathsf{Solve})$ be a TLP with solution space S and let $\mathtt{PKE} = (\mathtt{PKE}.\mathsf{Gen}, \mathtt{PKE}.\mathsf{Enc}, \mathtt{PKE}.\mathsf{Dec})$ be a PKE scheme. Figure 3 describes our construction of a TRE scheme. As we have already mentioned, the hardness parameter T is not necessary as input for Dec, hence we leave it out in the construction. Observe that correctness is directly implied by correctness of the PKE scheme and the TLP.

```
$\underline{\mathsf{Setup}(1^\lambda, T)}$
$(Z, s) \leftarrow \mathtt{TLP}.\mathsf{Gen}(T)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s)$
return $\mathsf{pp}_e := \mathsf{pk}, \mathsf{pp}_d := Z$

$\underline{\mathsf{Solve}(\mathsf{pp}_d)}$
$s \leftarrow \mathtt{TLP}.\mathsf{Solve}(\mathsf{pp}_d)$
return $s$

$\underline{\mathsf{Enc}(\mathsf{pp}_e, m)}$
return $c \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pp}_e, m)$

$\underline{\mathsf{Dec}(s, c)}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s)$
return $m \leftarrow \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}, c)$
```

Fig. 3. Construction of $\mathtt{TRE}$

Theorem 6. *If* $\mathtt{TLP} = (\mathtt{TLP}.\mathsf{Gen}, \mathtt{TLP}.\mathsf{Solve})$ *is secure time-lock puzzle with gap* ϵ *in the sense of Definition 2 and* $\mathtt{PKE} = (\mathtt{PKE}.\mathsf{Gen}, \mathtt{PKE}.\mathsf{Enc}, \mathtt{PKE}.\mathsf{Dec})$ *is a CPA secure encryption scheme according to Definition 16, then* $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Solve}, \mathsf{Enc}, \mathsf{Dec})$ *defined in Figure 3 is a secure timed-release encryption scheme with gap* $\underline{\epsilon} < \epsilon$ *in the sense of Definition 14.*

Proof. To prove security we define two games G_0 and G_1 and show that these are computationally indistinguishable.

Game 0. Game G_0 corresponds to original security experiment, where we use the Setup directly from our construction.

Game 1. In game G_1 we replace Setup with the alternative setup algorithm Setup' in which we sample s' independently at random from S and use it as randomness for $\mathtt{PKE}.\mathsf{Gen}$, i.e., run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; s')$.

Let $\tilde{T}_{\mathtt{TLP}}(\lambda)$ be the polynomial whose existence is guaranteed by the security of TLP. Let $\mathsf{poly}_{\mathtt{PKE}}(\lambda)$ be the fixed polynomial which bounds the time required to run $\mathtt{PKE}.\mathsf{Gen}$ and $\mathtt{PKE}.\mathsf{Enc}$. Set $\underline{T} := (\mathsf{poly}_{\mathtt{PKE}}(\lambda))^{1/\underline{\epsilon}}$. Set $\tilde{T}_{\mathtt{TRE}} := \max(\tilde{T}_{\mathtt{TLP}}, \underline{T})$.

Lemma 2. *From any polynomial-size adversary* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$*, where the depth of* $\mathcal{A}_\lambda$ *is bounded from above by* $T^{\underline{\epsilon}}(\lambda)$ *for some* $T(\cdot) \geq \tilde{T}_{\mathtt{TRE}}(\cdot)$ *we can construct a polynomial-size adversary* $\mathcal{B} = \{\mathcal{B}_\lambda\}_{\lambda \in \mathbb{N}}$*, where the depth of* $\mathcal{B}_\lambda$ *is bounded by* $T^\epsilon(\lambda)$ *with*

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

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

1. $\mathcal{B}_\lambda$ receives (Z, s) and begins to simulate the security experiment from Definition 14 by generating $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda, s)$.
2. Then it runs $\mathcal{A}_{1,\lambda}$ which yields $(m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_{1,\lambda}(\mathsf{pk}, Z)$.
3. $\mathcal{B}_\lambda$ picks $b \xleftarrow{\$} \{0, 1\}$ and computes $c \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pk}, m_b)$.
4. Finally, it runs $b' \leftarrow \mathcal{A}_{2,\lambda}(c, \mathsf{st})$ and returns the truth value of $b' = b$.

Note that if s is the solution of the puzzle Z, then $\mathcal{B}$ simulates G_0 perfectly. If s is random, then $\mathcal{B}$ simulates G_1 perfectly. Moreover, $\mathcal{B}$ meets the depth constraint:

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

Also $T(\cdot) \geq \tilde{T}_{\mathtt{TRE}}(\cdot) \geq \tilde{T}_{\mathtt{TLP}}(\cdot)$.

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

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

Lemma 3. *From any polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ we can construct a polynomial-size adversary $\mathcal{B}' = \{\mathcal{B}'_\lambda\}_{\lambda\in\mathbb{N}}$ such that*

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

The proof of this claim is straightforward. Notice that in G_1 we use fresh randomness which is independent of the puzzle Z. We construct $\mathcal{B}'_\lambda$ as follows:

1. $\mathcal{B}'_\lambda$ has pk as input and starts to simulate G_1 by running $(Z, s) \leftarrow \mathtt{TLP}.\mathsf{Gen}(T)$.
2. Then it runs adversary $(m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_{1,\lambda}(\mathsf{pk}, Z)$.
3. It outputs (m_0, m_1) to its experiment and receives c.
4. Finally it returns $b \leftarrow \mathcal{A}_{2,\lambda}(c, \mathsf{st})$.

Adversary $\mathcal{B}'$ simulates G_1 perfectly. Since $\mathcal{A}$ has polynomially-bounded size, this proves the claim. By combining Lemma 1 and Lemma 2 we obtain following:

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

which concludes the proof. □

4.2 Sequential TRE

In the sequel let $\mathtt{sTLP} = (\mathtt{sTLP}.\mathsf{Gen}, \mathtt{sTLP}.\mathsf{Solve})$ be a sequential TLP in the sense of Definition 8 and let $\mathtt{PKE} = (\mathtt{PKE}.\mathsf{Gen}, \mathtt{PKE}.\mathsf{Enc}, \mathtt{PKE}.\mathsf{Dec})$ be a PKE scheme. Let $\mathsf{F} : \mathbb{N} \times S \to Y$ be a function that maps the hardness parameter space $\mathbb{N}$ and the solution space S of $\mathtt{sTLP}$ to the randomness space of algorithm $\mathtt{PKE}.\mathsf{Gen}$. Our constructions of a sequential TRE scheme $\mathtt{TRE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ is given in Figure 4. Note that correctness of the scheme is directly implied by correctness of the PKE scheme and the sequential TLP.

Theorem 7. *If* $\mathtt{sTLP} = (\mathtt{sTLP}.\mathsf{Gen}, \mathtt{sTLP}.\mathsf{Solve})$ *is a secure sequential time-lock puzzle with gap ϵ w.r.t. function* 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 4 is a secure sequential timed-release encryption with gap $\underline{\epsilon} < \epsilon$.*

Proof. To prove security we define a sequence of games.

```
$\underline{\mathsf{Setup}(1^\lambda, (T_i)_{i\in[n]})}$
$(Z_i, s_i)_{i\in[n]} \leftarrow \mathsf{sTLP.Gen}((T_i)_{i\in[n]})$
$((\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(T_i, s_i)))_{i\in[n]}$
return $(\mathsf{pp}_{e,i} := \mathsf{pk}_i, \mathsf{pp}_{d,i} := Z_i)_{i\in[n]}$

$\underline{\mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})}$
$s_i \leftarrow \mathsf{sTLP.Solve}(\mathsf{pp}_{d,i}, s_{i-1})$
return $s_i$

$\underline{\mathsf{Enc}(\mathsf{pp}_{e,i}, m)}$
return $c \leftarrow \mathtt{PKE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, m)$

$\underline{\mathsf{Dec}(T_i, s_i, c)}$
$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(T_i, s_i))$
return $m \leftarrow \mathtt{PKE}.\mathsf{Dec}(\mathsf{sk}_i, c)$
```

Fig. 4. Construction of sequential TRE

Game 0. Game G_0 is the original security experiment with scheme TRE.

Game 1. This game is identical to G_0, except that at the beginning of game 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 G_0.

Lemma 4. *We have that* $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{G}_0} = n \cdot \mathbf{Adv}_{\mathcal{A}}^{\mathsf{G}_1}$.

This lemma is proven using a standard argument.

Game 2. In Game G_2 we replace Setup with the alternative setup algorithm Setup', which takes as input i^*. For all $j \in [n] \setminus \{i^*\}$ we produce keys pk_j using $\mathsf{F}(T_j, s_j)$. The remaining key pk_{i^*} is generated using fresh randomness sampled uniformly from the image of the function F.

Let $\tilde{T}_{\mathtt{sTLP}}(\lambda)$ be the polynomial whose existence is guaranteed by the security of sTLP. Let $\mathsf{poly}_{\mathtt{PKE}}(\lambda)$ be the fixed polynomial which bounds the time required to run $\mathtt{PKE.Gen}$ n-times and to run $\mathtt{PKE.Enc}$ once. Set $\underline{T} := (\mathsf{poly}_{\mathtt{PKE}}(\lambda))^{1/\underline{\epsilon}}$. Set $\tilde{T}_{\mathtt{TRE}} := \max(\tilde{T}_{\mathtt{sTLP}}, \underline{T})$.

Lemma 5. *For any n which is polynomial in λ, for any set of polynomials $(T_j(\cdot))_{j\in[n]}$ fulfilling that $\forall j \in [n]$ holds $T_j(\cdot) \geq \tilde{T}_{\mathtt{TRE}}(\cdot)$, for any $i \in [n]$, from any polynomial-size adversary $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda})\}_{\lambda\in\mathbb{N}}$, where the depth of $(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda})$ is bounded from above by $T_i^{\underline{\epsilon}}(\lambda)$, we can construct a polynomial-size adversary $\mathcal{B}_{i^*} = \{\mathcal{B}_{i^*,\lambda}\}_{\lambda\in\mathbb{N}}$ whose depth is bounded from above by $T_i^{\epsilon}(\lambda)$ such that*

$$\mathbf{Adv}_{\mathcal{B}_{i^*}}^{\mathsf{sTLP}} \geq \left|\Pr[\mathsf{G}_1 = 1] - \Pr[\mathsf{G}_2 = 1]\right|.$$

To prove this claim we construct an adversary $\mathcal{B}_{i^*,\lambda}$ as follows.

1. $\mathcal{B}_{i^*,\lambda}$ receives $(Z_j, y_j)_{j\in[n]}$ and simulates the game by running $((\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathtt{PKE}.\mathsf{Gen}(1^\lambda; y_j))_{j\in[n]}$.
2. Then it runs adversary $(i, m_0, m_1, \mathsf{st}) \xleftarrow{\$} \mathcal{A}_{1,\lambda}((\mathsf{pk}_j, Z_j)_{j\in[n]}, \mathsf{st})$.
3. If $i \neq i^*$, then it returns a random bit $b' \xleftarrow{\$} \{0,1\}$.

4. Otherwise it picks $b \xleftarrow{\$} \{0,1\}$ and compute $c \leftarrow \texttt{PKE}.\mathsf{Enc}(\mathsf{pk}_i, m_b)$.
5. It runs $b' \leftarrow \mathcal{A}_{2,\lambda}(c, \mathsf{st})$ and returns the truth value of $b' = b$.

If $y_j = \mathsf{F}(T_j, s_j)$ for all $j \in [n]$, then $\mathcal{B}_{i^*}$ simulates Game G_1 perfectly. If the y_{i^*} is random, then it simulates Game G_2 perfectly. Therefore we obtain

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

and thus

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

Moreover, $\mathcal{B}_{i^*}$ fulfils the depth constraint:

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

Also $T_i(\cdot) \geq \tilde{T}_{\texttt{TRE}}(\cdot) \geq \tilde{T}_{\texttt{sTLP}}(\cdot)$ as required.

Lemma 6. *For any n which is polynomial in λ, for any set of polynomials $(T_j(\cdot))_{j\in[n]}$ fulfilling that $\forall j \in [n]$ holds $T_j(\cdot) \geq \tilde{T}_{\texttt{TRE}}(\cdot)$, for any $i \in [n]$, from any polynomial-size adversary $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda})_{\lambda\in\mathbb{N}}$ we can construct a polynomial-size adversary $\mathcal{B}' = \{\mathcal{B}'_\lambda\}_{\lambda\in\mathbb{N}}$ such that*

$$\left|\mathbf{Adv}^{\texttt{PKE}}_{\mathcal{B}'} = \Pr[\mathsf{G}_2 = 1] - \frac{1}{2}\right|.$$

The proof is essentially identical to the corresponding step from proof of Theorem 6, adopted to the sequential setting. We construct $\mathcal{B}'_\lambda$ as follows:

1. $\mathcal{B}'_\lambda$ receives as input pk as input and starts to simulate G_2 by sampling $i^* \xleftarrow{\$} [n]$ uniformly at random and running $(Z_j, s_j)_{j\in[n]} \leftarrow \texttt{sTLP}.\mathsf{Gen}((T_j)_{j\in[n]})$.
2. For all $j \in [n] \setminus \{i^*\}$ it sets $\mathsf{pk}_j := \texttt{PKE}.\mathsf{Gen}(1^\lambda; \mathsf{F}(T_j, s_j))$. The i^*-th public key is defined as $\mathsf{pk}_{i^*} := \mathsf{pk}$.
3. $\mathcal{B}'_\lambda$ runs adversary $(i, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}_{1,\lambda}((\mathsf{pk}_j, Z_j)_{j\in[n]}, \mathsf{st})$.
4. If $i \neq i^*$ it samples and outputs $b' \xleftarrow{\$} \{0,1\}$.
5. Else it sends (m_0, m_1) to its experiment and receives c.
6. Finally, it returns $b \leftarrow \mathcal{A}_{2,\lambda}(c, \mathsf{st})$.

Note that adversary $\mathcal{B}'$ simulates G_2 perfectly, which yields Lemma 6.
Finally, combining Lemmas 4-6 we obtain

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

which concludes the proof. □

4.3 Integrating Timed-Release Features into Functional Encryption

In this section, we connect sequential timed-release features with functional encryption (FE) [8,29]. We recap the FE definition, its correctness, and security notion in Appendix B where we also discuss one concrete FE variant, namely identity-based encryption (IBE) [6].

Timed-Release Functional Encryption. We introduce the notion of a (sequential) timed-release functional encryption (TRFE) scheme. The basic idea is that in such a scheme, similarly to an FE scheme, there is a public key pk used for encryption of any message x and a master secret key msk which is associated to a class of functions $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$. In contrast to an FE scheme, however, in a TRFE scheme, msk can be used to generate decryption keys for a function $f \in \mathcal{F}$ which is associated to a time hardness parameter T_i (and, hence, to its solution s_i). Decryption takes the associated decryption key dk_i, the solution s_i, a function $f \in \mathcal{F}$, and a ciphertext to message x and outputs $f(x)$. Security-wise, an adversary is allowed to query *any* secret function key for any public encryption parameter associated to T_i as long as its solution s_i is not retrievable.

Definition 17. *A (sequential) TRFE scheme* TRFE *for a class of functions* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ *consists of the PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$*:*

- $(\mathsf{pk}, \mathsf{msk}, \mathsf{PP} := (\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}, (T_j)_{j\in[n]})$*, on input security parameter* 1^λ*, class of functions* $\mathcal{F}$*, and time-hardness parameters* $(T_j)_{j\in[n]}$ *with* $T_1 < \ldots < T_n$*, for* $n \in \mathbb{N}$*, outputs public key* pk*, main secret key* msk*, and public encryption and decryption parameters* $(\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]}$*. We require that* Setup *runs in time at most* $\mathsf{poly}((\log T_j)_{j\in[n]}, \lambda)$*.*
- $\mathsf{dk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, (\mathsf{pp}_{e,j})_{j\in[n]}, f, i)$*, on input main secret key* msk*, public encryption parameters* $(\mathsf{pp}_{e,j})_{j\in[n]}$*, function* $f \in \mathcal{F}$*, and index* $i \in [n]$*, outputs decryption key* dk_i*.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$*, on input public key* pk *and message* $x \in \mathcal{X}$*, outputs 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 from a previous iteration* s_{i-1}*, where* $s_0 = \perp$*, outputs solution* s_i*. We require that* Solve *runs in time at most* $(T_i - T_{i-1}) \cdot \mathsf{poly}(\lambda)$*, where* T_i *and* T_{i-1} *are the associated hardness parameters for* s_i *and* s_{i-1}*, respectively.*
- $f(x') \leftarrow \mathsf{Dec}(\mathsf{dk}_i, T_i, s_i, c)$ *is a deterministic algorithm which on input decryption key* dk_i*, time-hardness parameter* T_i*, solution* s_i*, function* f*, and ciphertext* c*, outputs* $f(x') \in \mathcal{Y} \cup \{\perp\}$*.*

We say a (sequential) TRFE scheme TRFE *is* correct *if for all* $\lambda, n \in \mathbb{N}$*, for any class of functions* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$*, for all sets of time-hardness parameters* $(T_j)_{j\in[n]}$ *such* *that $T_1 < \ldots < T_n$, for all $f \in \mathcal{F}$, $i \in [n]$, $x \in \mathcal{X}$, it holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{dk}_i, T_i, s_i, c) = f(x) \;:\; \begin{array}{r} (\mathsf{pk}, \mathsf{msk}, \mathsf{PP}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{F}, (T_j)_{j\in[n]}) \\ \mathsf{dk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, (\mathsf{pp}_{e,j})_{j\in[n]}, f, i) \\ s_i \leftarrow \mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1}), s_0 = \bot \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, x) \end{array}\right] = 1.$$

Definition 18. *A (sequential) TRFE scheme* TRFE *is* CPA secure with gap $0 < \epsilon < 1$ *if for all polynomials n in λ there exists a polynomial $\tilde{T}(\cdot)$ such that for all sets of polynomials $(T_j)_{j\in[n]}$ fulfilling that $\forall j \in [n] : T_j(\cdot) \geq \tilde{T}(\cdot)$, for all $i \in [n]$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda = (\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}, \mathcal{A}_{3,\lambda})\}_{\lambda\in\mathbb{N}}$, for all $\lambda \in \mathbb{N}$, we have*

$$\mathbf{Adv}^{\mathsf{TRFE}}_{\mathcal{A}} = \left|\Pr\left[b = b' \colon \begin{array}{r} (\mathsf{pk}, \mathsf{msk}, \mathsf{PP}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}, (T_j)_{j\in[n]}) \\ (i^*, \mathsf{st}) \leftarrow \mathcal{A}_{1,\lambda}(\mathsf{pk}, \mathsf{PP}) \\ (x_0, x_1, \mathsf{st}) \leftarrow \mathcal{A}_{2,\lambda}^{\mathsf{KeyGen}(\mathsf{msk}, (\mathsf{pp}_{e,j})_{j\in[n]}, \cdot, \cdot)}(\mathsf{st}) \\ b \leftarrow \{0,1\}, c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b) \\ b' \leftarrow \mathcal{A}_{3,\lambda}^{\mathsf{KeyGen}(\mathsf{msk}, (\mathsf{pp}_{e,j})_{j\in[n]}, \cdot, \cdot)}(\mathsf{st}, c^*) \end{array}\right] - \frac{1}{2}\right| = \mathsf{negl}(\lambda),$$

for negligible function negl*, where we require that $|x_0| = |x_1|$ and $(\mathcal{A}_2, \mathcal{A}_3)$ only query* KeyGen *with functions $f \in \mathcal{F}$ and indexes $i \in [i^* - 1]$ such that $f(x_0) = f(x_1)$. We require that $\mathcal{A}_\lambda$ consist of three circuits with total depth at most $T^{\epsilon}_{i^*}(\lambda)$ (i.e., the total depth is the sum of the depth of $\mathcal{A}_{1,\lambda}$, $\mathcal{A}_{2,\lambda}$, and $\mathcal{A}_{3,\lambda}$).*

Construction of TRFE. Let $\mathtt{TRE} = (\mathtt{TRE}.\mathsf{Setup}, \mathtt{TRE}.\mathsf{Solve}, \mathtt{TRE}.\mathsf{Enc}, \mathtt{TRE}.\mathsf{Dec})$ be a (sequential) TRE scheme and $\mathtt{FE} = (\mathtt{FE}.\mathsf{Gen}, \mathtt{FE}.\mathsf{KeyGen}, \mathtt{FE}.\mathsf{Enc}, \mathtt{FE}.\mathsf{Dec})$ be an FE scheme. We construct a TRFE scheme $\mathsf{TRFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ as given in Figure 5. Let the message space of TRE be the functional-secret-key space of FE which is the output of FE.KeyGen. Furthermore, we need that all functional secret keys for function $f \in \mathcal{F}$ and any main secret key of FE are of equal length.

Theorem 8. *If* $\mathtt{TRE} = (\mathtt{TRE}.\mathsf{Setup}, \mathtt{TRE}.\mathsf{Solve}, \mathtt{TRE}.\mathsf{Enc}, \mathtt{TRE}.\mathsf{Dec})$ *is a secure TRE scheme with gap ϵ in the sense of Definition 14 and* $\mathtt{FE} = (\mathtt{FE}.\mathsf{Gen}, \mathtt{FE}.\mathsf{KeyGen}, \mathtt{FE}.\mathsf{Enc}, \mathtt{FE}.\mathsf{Dec})$ *is a CPA-secure FE scheme in the sense of Definition 20, then* $\mathsf{TRFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Solve}, \mathsf{Dec})$ *defined in Figure 5 is a CPA-secure TRFE scheme with gap $\underline{\epsilon} \leq \epsilon$ in the sense of Definition 18. Concretely, for any successful polynomial-size adversary $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}, \mathcal{A}_{3,\lambda})\}_{\lambda\in\mathbb{N}}$ on the CPA-security of* TRFE *with success probability $\varepsilon_{\mathcal{A}_{\mathsf{TRFE}}}$, we construct a successful polynomial-size adversary $\mathcal{B}$ on the security of* TRE *with probability $\varepsilon_{\mathtt{TRE}}/q \geq \varepsilon_{\mathcal{A}_{\mathsf{TRFE}}}$, with $q = \sum_{i=1}^{n} q_i$, for number of* KeyGen*-queries q_i for index $i \in [n]$, or on the CPA-security of* FE *with probability $\varepsilon_{\mathtt{FE}} \geq \varepsilon_{\mathcal{A}_{\mathsf{TRFE}}}$.*

Proof. We proceed in games G_0, G_1, G_2 where we show that all those games are computationally indistinguishable. G_0 reflects the CPA security experiment as given in Definition 18. In Game G_1, all KeyGen-queries for index i with $i^* \leq i \leq n$ are answered

```
$\underline{\mathsf{Gen}(1^\lambda, \mathcal{F}, (T_j)_{j\in[n]})}$
$(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$
$(\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]} \leftarrow \mathtt{TRE}.\mathsf{Setup}(1^\lambda, (T_j)_{j\in[n]})$
return $(\mathsf{pk}, \mathsf{msk}, (\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]})$

$\underline{\mathsf{Enc}(\mathsf{pk}, x)}$
return $c \leftarrow \mathtt{FE}.\mathsf{Enc}(\mathsf{pk}, x)$

$\underline{\mathsf{KeyGen}(\mathsf{msk}, (\mathsf{pp}_{e,j})_{j\in[n]}, f, i)}$
$\mathsf{sk}_f \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\mathsf{msk}, f)$
$c_i \leftarrow \mathtt{TRE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, \mathsf{sk}_f)$
return $\mathsf{dk}_i := c_i$

$\underline{\mathsf{Dec}(\mathsf{dk}_i, T_i, s_i, c)}$
$c_i := \mathsf{dk}_i$
$\mathsf{sk}_f := \mathtt{TRE}.\mathsf{Dec}(T_i, s_i, c_i)$
return $f(x) := \mathtt{FE}.\mathsf{Dec}(\mathsf{sk}_f, c)$

$\underline{\mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})}$
return $s_i := \mathtt{TRE}.\mathsf{Solve}(\mathsf{pp}_{d,i}, s_{i-1})$
```

Fig. 5. Construction of TRFE.

running $\mathsf{TRFE}.\mathsf{KeyGen}(\widetilde{\mathsf{msk}}, \cdot, \cdot, i)$, for fresh $(\cdot, \widetilde{\mathsf{msk}}, \cdot) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$. In Game G_2, the challenge ciphertext is independent of the bit b.

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

Lemma 7. *Let q_i be the number of* KeyGen*-queries for index i with $i^* \leq i \leq n$. For any successful polynomial-size adversary $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}, \mathcal{A}_{3,\lambda})\}_{\lambda\in\mathbb{N}}$ on the CPA-security of* TRFE *with success probability $\varepsilon_{\mathcal{A}}$ and bounded depth at most $T^{\epsilon}_{i^*}(\lambda)$, we construct a successful polynomial-size adversary $\mathcal{B}$ on the security of* TRE *with success probability $\varepsilon_{\mathcal{B}}/(q \cdot n) \geq \varepsilon_{\mathcal{A}}$.*

Proof. This Lemma can be shown by a hybrid argument where we go over all indexes i with $i^* \leq i \leq n$ and then go over all q_i queries for such an index in a step-by-step fashion. Since the proof is always the same depending on the index and the actual query number for that index, we show the proof for the j-th query (with $j \in [q_i]$) for index $i \in [n]$ and argue that this holds for all indexes i. For index i and query j, we construct an efficient hybrid between Game $0.i.j-1$ and Game $0.i.j$ with distinguisher $\mathcal{B}$ on the TRE security as follows:

1. $\mathcal{B}$ receives $(\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_{j\in[n]}$ for time hardness parameters $(T_j)_{j\in[n]}$ as input and runs $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$ and $(\cdot, \widetilde{\mathsf{msk}}, \cdot) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$. Then, $\mathcal{A}_1$ is started on $(\mathsf{pk}, (\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_j)$ and $\mathcal{A}_1$ outputs i^*. If $i^* \notin [n]$, $\mathcal{B}$ outputs $b \leftarrow \{0, 1\}$.
2. $\mathcal{A}_2$ is started by $\mathcal{B}$. For secret key queries of $\mathcal{A}_2$ (and later also of $\mathcal{A}_3$) on function f and index $i' < i^*$ or $i' > i$, $\mathcal{B}$ returns $\mathsf{KeyGen}(\mathsf{msk}, f, i')$; for all queries for index i' with $i^* \leq i' < i$, compute $\widetilde{\mathsf{sk}}_f \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\widetilde{\mathsf{msk}}, f, i')$; for index i' and j'-th query with $i' = i$ and $j' < j$, compute $\widetilde{\mathsf{sk}}_f \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\widetilde{\mathsf{msk}}, f, i)$; for index i' and j'-th query with $i' = i$ and $j' = j$ compute $\mathsf{sk}_0 \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\mathsf{msk}, f, i')$ and $\mathsf{sk}_1 \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\widetilde{\mathsf{msk}}, f, i')$, and send $(i, \mathsf{sk}_f, \widetilde{\mathsf{sk}}_f)$ to $\mathcal{B}$'s TRE challenger where the returning ciphertext $c^* \leftarrow \mathtt{TRE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, \mathsf{sk}_{b^*})$, for unknown $b^* \leftarrow \{0, 1\}$, is forwarded as secret key to $\mathcal{A}$; finally, for the j'-th query on index i' with $i' = i$ and $j' > j$, compute $\mathsf{sk}_f \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\mathsf{msk}, f, i')$.

3. At some point, $\mathcal{A}_2$ outputs (x_0, x_1) where $\mathcal{B}$ returns $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, x_b)$ to $\mathcal{A}_3$, for uniform $b \leftarrow \{0,1\}$.
4. Eventually, $\mathcal{A}_3$ outputs a guess b'. If $b' = b$, $\mathcal{B}$ returns 1 else returns 0 to $\mathcal{B}$'s TRE challenger.

See that if $\mathcal{A}_2$ and $\mathcal{A}_3$ query KeyGen for indexes $i < i^*$ with $f(x_0) = f(x_1)$ and $b^* = 0$, then $\mathcal{B}$ simulates Game $0.i.j-1$, otherwise, if $\mathcal{A}_2$ and $\mathcal{A}_3$ query KeyGen for indexes $i < i^*$ with $f(x_0) = f(x_1)$ and $b^* = 1$, $\mathcal{B}$ simulates Game $0.i.j$.

Note that we arrived at a game where all $\mathtt{TRE}$-encryptions for key queries for index $i \geq i^*$ contain msk-independent secret keys.

Lemma 8. *For any successful polynomial-size $\mathcal{A} = \{(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}, \mathcal{A}_{3,\lambda})\}_{\lambda\in\mathbb{N}}$ on the CPA-security of* TRFE *with success probability $\varepsilon_{\mathcal{A}}$ and bounded depth at most $T^{\epsilon}_{i^*}(\lambda)$, we construct a successful polynomial-size adversary $\mathcal{B}$ on the security of* $\mathtt{FE}$ *with success probability $\varepsilon_{\mathcal{B}} \geq \varepsilon_{\mathcal{A}}$.*

Proof. We first construct $\mathcal{B}$ as follows:

1. $\mathcal{B}$ receives pk as input and computes $(\cdot, \widetilde{\mathsf{msk}}, \cdot) \leftarrow \mathtt{FE}.\mathsf{Gen}(1^\lambda, \mathcal{F})$. $\mathcal{A}_1$ is started on $(\mathsf{pk}, (\mathsf{pp}_{e,j}, \mathsf{pp}_{d,j})_j)$, for $(\mathsf{pp}_{e,i}, \mathsf{pp}_{d,i})_i \leftarrow \mathtt{TRE}.\mathsf{Setup}(1^\lambda, (T_i)_i)$, and outputs i^*.
2. For any query by $\mathcal{A}_2$ or $\mathcal{A}_3$ on f for index i with $i < i^*$ to KeyGen, $\mathcal{B}$ forwards f to $\mathcal{B}$'s KeyGen-oracle, encrypts the result sk_f under $\mathsf{pp}_{e,i}$ computing $c_i \leftarrow \mathtt{TRE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, \mathsf{sk}_f)$ and returns c_i. For indexes i with $i^* \leq i \leq n$, $\mathcal{B}$ computes $\widetilde{\mathsf{sk}}_f \leftarrow \mathtt{FE}.\mathsf{KeyGen}(\widetilde{\mathsf{msk}}, f, i)$, $c_i \leftarrow \mathtt{TRE}.\mathsf{Enc}(\mathsf{pp}_{e,i}, \widetilde{\mathsf{sk}}_f)$, and returns c_i.
3. $\mathcal{B}$ runs $\mathcal{A}_2$ to receive (x_0, x_1), forwards (x_0, x_1) to $\mathcal{B}$'s challenger, receives c^* and starts $\mathcal{A}_3$ with c^*.
4. Finally, $\mathcal{B}$ receives b' which is forwarded to its challenger.

$\mathcal{B}$ simulates the experiment perfectly. See that if $\mathcal{A}$ is a successful adversary with probability $\varepsilon_{\mathcal{A}}$, then $\mathcal{B}$ is a successful adversary in the CPA-security experiment with probability $\varepsilon_{\mathcal{B}} \geq \varepsilon_{\mathcal{A}}$.

We arrived at a game where the adversary $\mathcal{A}$ receives a challenge ciphertext that is independent of b which shows the Theorem. □

Application to locked-key IBE. With TRFE, we are able to lock secret keys of an IBE scheme with a sequential 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 sequentially. This, for example, enables an IBE central authority to produce all secret keys in the beginning and afterwards go off-line.

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

1. Abadi, A., Kiayias, A.: Multi-instance publicly verifiable time-lock puzzle and its applications. Financial Cryptography and Data Security
2. Applebaum, B.: Garbled circuits as randomized encodings of functions: a primer. In: Tutorials on the Foundations of Cryptography
3. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. Comput. Complex.
4. Ball, M., Dachman-Soled, D., Kulkarni, M., Lin, H., Malkin, T.: Non-malleable codes against bounded polynomial time tampering. Cryptology ePrint Archive, Report 2018/1015, `https://eprint.iacr.org/2018/1015`
5. Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., Waters, B.: Time-lock puzzles from randomized encodings. In: ITCS 2016: 7th Conference on Innovations in Theoretical Computer Science
6. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Advances in Cryptology – CRYPTO 2001
7. Boneh, D., Naor, M.: Timed commitments. In: Advances in Cryptology – CRYPTO 2000
8. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC 2011: 8th Theory of Cryptography Conference
9. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In: TCC 2019: 17th Theory of Cryptography Conference, Part II
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: ITCS 2012: 3rd Innovations in Theoretical Computer Science
11. Cheon, J.H., Hopper, N., Kim, Y., Osipkov, I.: Provably secure timed-release public key encryption. ACM Trans. Inf. Syst. Secur.
12. Di Crescenzo, G., Ostrovsky, R., Rajagopalan, S.: Conditional oblivious transfer and timed-release encryption. In: Advances in Cryptology – EUROCRYPT'99
13. Dwork, C., Naor, M.: Zaps and their applications. In: 41st Annual Symposium on Foundations of Computer Science
14. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Advances in Cryptology – CRYPTO'84
15. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779, `https://eprint.iacr.org/2020/779`
16. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Advances in Cryptology – CRYPTO 2018, Part II
17. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: 41st Annual ACM Symposium on Theory of Computing
18. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: 45th Annual ACM Symposium on Theory of Computing
19. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st Annual Symposium on Foundations of Computer Science
20. Katz, J., Loss, J., Xu, J.: On the security of time-lock puzzles and timed commitments. In: TCC 2020: 18th Theory of Cryptography Conference, Part III
21. Koppula, V., Lewko, A.B., Waters, B.: Indistinguishability obfuscation for turing machines with unbounded memory. In: 47th Annual ACM Symposium on Theory of Computing

22. Lin, H., Pass, R., Soni, P.: Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In: 58th Annual Symposium on Foundations of Computer Science
23. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Designs, Codes and Cryptography
24. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: 44th Annual ACM Symposium on Theory of Computing
25. Mahmoody, M., Moran, T., Vadhan, S.P.: Time-lock puzzles in the random oracle model. In: Advances in Cryptology – CRYPTO 2011
26. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Advances in Cryptology – CRYPTO 2019, Part I
27. May, T.C.: Timed-release crypto. Tech. rep.
28. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography
29. O'Neill, A.: Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, `https://eprint.iacr.org/2010/556`
30. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Advances in Cryptology – EUROCRYPT'99
31. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep.
32. Shamir, A.: How to share a secret. Commun. ACM
33. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Advances in Cryptology – EUROCRYPT'97
34. Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, `https://eprint.iacr.org/2004/332`
35. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable timed signatures made practical. In: to appear at ACM CCS 2020, `https://verifiable-timed-signatures.github.io/web/assets/paper.pdf`
36. Unruh, D.: Revocable quantum timed-release encryption. In: Advances in Cryptology – EUROCRYPT 2014

Appendix

A On the Necessity of the Gap Sequential Squaring Assumption

One might ask why the following seemingly simple solution does not yield a secure sequential TLP:

- Generate a set of (non-sequential) puzzles $(Z_1, s_1), ..., (Z_n, s_n)$, such that the delay parameter for puzzle i is $T_i - T_{i-1}$.
- Let $(\mathsf{Enc}, \mathsf{Dec})$ be some CPA-secure symmetric encryption scheme.
- Publish $(Z_1, (\mathsf{Enc}_{s_{i-1}}(Z_i))_{i=2}^{n})$.

Unfortunately, this approach does not work (see also page 16). Concretely, suppose we have an adversary which has sufficient running time to solve $n-1$ puzzles, and then successfully attacks the n-th puzzle (say, with success probability 1, for instance). Now note that we cannot use the CPA security of any of the first $n-1$ encryptions to "hide" any intermediate puzzle, because the adversary has enough time to notice this (as it has enough running time to solve all the first $n-1$ puzzles). However, then, since we cannot use the CPA security as an argument in the proof, we can equivalently consider the first $n-1$ encryption as completely insecure.

But if we have no security guarantees for the first $n-1$ encryptions, this means that we also cannot argue that the adversary cannot obtain the n-th puzzle instance Z_n quickly, without solving $n-1$ prior instances, because it could simply "break" the $(n-1)$-th encryption to obtain Z_n (which might be very quick, e.g., in just a few computational steps, because we cannot argue that the scheme is secure in the sense of CPA or some other notion). And then an adversary with sufficient running time to solve $n-1$ puzzles can simply solve only the last puzzle using the standard Solve algorithm.

So in conclusion, we do not claim that the construction is insecure, but only that CPA security or a similar notion seems not sufficient, as we cannot perform a reduction to the CPA security in the usual way. It might be possible to prove security under a non-standard assumption (e.g., by essentially assuming that the proposed construction is secure), however, this would also be an additional assumption, which we want to avoid.

B Functional Encryption

Definition 19. *A functional encryption scheme* FE *for a class of functions* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ *consists of four PPT algorithms* $(\mathsf{Gen}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F})$*, on input security parameter* 1^λ *and class of functions* $\mathcal{F}$*, outputs public key* pk *and master secret key* msk.
- $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$*, on input* msk *and function* $f \in \mathcal{F}$*, outputs secret key* sk_f *for* f.
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$*, on input* pk *and message* $x \in \mathcal{X}$*, outputs ciphertext* c *for* x.

– $f(x) \leftarrow \mathsf{Dec}(\mathsf{sk}_f, c)$ *is a deterministic algorithm which on input* sk_f *and c, outputs* $f(x) \in \mathcal{Y} \cup \{\perp\}$.

We say an FE scheme FE *is correct if for all* $\lambda \in \mathbb{N}$*, for any* $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$*, for any* $f \in \mathcal{F}$*, for any* $x \in \mathcal{X}$*, 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 20. *An FE scheme* FE *is CPA secure if for all non-uniform PPT adversaries* $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$*, we have*

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

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

Example: identity-based encryption. We recall that in identity-based encryption (IBE) [6], 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 19 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 secret key $\mathsf{sk}_{f_{id^*}}$ for identity id^* is generated with respect to f_{id^*} defined as:

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

C Applications: Simpler and More Efficient Instantiations

Subsequently, we discuss the applications in [26] when we use our (homomorphic) TRE approach in contrast to HTLPs of MT19. All the following application have in common that they 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 [26] 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 Setup is run. In all of this applications we are able to use our TRE approach *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 [26], 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 Solve only once. The output s of Solve allows to obtain the secret key which is then used to decrypt all ciphertexts that have been produced using corresponding pp_e. Therefore, if we encrypt all votes using the same 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 TRE scheme guarantees that all votes remain hidden during the whole voting phase. In the e-voting protocol proposed in [26], we have to wait until the voting phase is finished and then we can combine puzzles from voting phase to m resulting puzzles (one per candidate where votes are encoded as 0 and 1 respectively). Then, these m puzzles can be solved, which requires at least time T and solving m puzzles in parallel. Hence, it requires time T after the voting phase is over to be able to announce the results. This is in contrast to what we can do with our TRE, in which we can encrypt the respective encoding of the candidate, e.g., $i \in [m]$ directly, and can start to solve a *single* puzzle immediately after 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.

The approach proposed in [26] relies on HTLPs and their protocol consist of three phases: Setup, Coin Flipping and Announcement of the result. Similarly to the e-voting protocol, one is only able to start solving the puzzle in the last phase and hence obtains the results after time T. We are able to avoid this problem, by using our TRE approach, 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 the opening phase is equal to number of bidders who went offline. In [26] this problem was resolved by using HLTPs. Again, this solution has the same issues as the ones discussed above and can be avoided using our 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 in [26] consists of four phases - Setup, Key Generation, Signing and Aggregation, and combines aggregate signatures from RSA with multiplicatively homomorphic time-lock puzzles with a setup that allows producing puzzles for multiple hardness parameters. We remark that this type of time-lock puzzles are in some sense equivalent to our sequential timed-release encryption.[5] The protocol runs in ℓ-rounds and 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 have not broadcasted their puzzles in any round, the parties will homomorphically evaluate puzzles from the 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 sequential TRE, we are able to obtain result in time $T_{\ell-i+1}$ after the setup was executed. Moreover, we can combine sequential TRE with an arbitrary aggregate signature scheme, because we do not need to perform any homomorphic evaluation.

[5] Though they only discuss them informally in [26] and as mentioned in Section 1 it seems that it is not possible to prove it secure as it is proposed.