Poster: Tight Short-Lived Signatures

Arup Mondal
Ashoka University
Sonipat, Haryana, India
arup.mondal_phd19@ashoka.edu.in

Ruthu Hulikal Rooparaghunath
Vrije Universiteit
Amsterdam, The Netherlands
r.rooparaghunath@student.vu.nl

Debayan Gupta
Ashoka University
Sonipat, Haryana, India
debayan.gupta@ashoka.edu.in

***Abstract*—A Time-lock puzzle (TLP) sends information into the future: a predetermined number of sequential computations must occur (i.e., a predetermined amount of time must pass) to retrieve the information, regardless of parallelization. Buoyed by the excitement around secure decentralized applications and cryptocurrencies, the last decade has witnessed numerous constructions of TLP variants and related applications (e.g., cost-efficient blockchain designs, randomness beacons, e-voting, etc.).**

In this poster, we first extend the notion of TLP by formally defining the "time-lock public key encryption" (TLPKE) scheme. Next, we introduce and construct a "tight short-lived signatures" scheme using our TLPKE. Furthermore, to test the validity of our proposed schemes, we do a proof-of-concept implementation and run detailed simulations.

***Index Terms*—Time-Lock Puzzle, Short-Lived Signatures.**

I. Introduction

A short-lived signature (SLS) provides a verifier with two possibilities: either a generated signature σ on m is correct, or a user has expended a minimum predetermined amount of sequential work (T steps or time) to forge the signature. In other words, the signatures created remain valid for a short period of time T. Formally, we define the unforgeability period as the time starting from when a signer creates a signature for a message using their private signing information (or key) and the `Sign` algorithm. Once the unforgeability period T has elapsed, anyone can compute a forged signature using some public signing information and the `ForgeSign` algorithm.

Recall that any party can compute a forged signature (using `ForgeSign`) after the unforgeability period T has passed. However, there is no guarantee that a party cannot generate a forged signature in *advance*. To ensure this, we use the same model used in [1]. The signature incorporates a random beacon value to ensure it was not created before a specific time T_0. Suppose a verifier observes the signature within $\hat{T}$ units of time after T_0. In this case, they will believe it is a valid signature if $\hat{T} < T$ because it would be impossible to have forged the signature within that time period. Once $\hat{T} \geq T$, the signature is no longer convincing as it may have been constructed through forgery.

Brief Concurrent Work: Recently, Arun et al. [1] studied the variants notion of short-lived cryptographic primitives, i.e., short-lived proofs and signatures. Similar to our work, they make use of sequentially-ordered computations (T-sequential computation) as a means to enforce time delay during which signatures are unforgeable but become forgeable afterward ($(1+c)\cdot T$-sequential computations). In this work, we use the same models as used in [1], however, we define and construct **tight** short-lived signatures, where the forged signatures can be generated in time not much more than sequentially bound T. In other words, tight short-lived signatures ensure that forged signatures can be generated in **exactly** T sequential computations.

TABLE I: Complexity comparison of SLS schemes.

Paper	Setup & Sign	Forge Sign	Verify	Tight
Arun et al. [1]	$\mathsf{poly}(\lambda)$	$O((1+c)\cdot T)$	VDF ([2], [3])	No
Algorithm	$\mathsf{poly}(\lambda)$	$O(T)$	$O(1)$	Yes

Short-lived cryptographic primitives have many real-life use cases; we refer to [1] for a detailed discussion of its applications. *Our main contributions are summarised as follows:*

- First, we extend the time-lock puzzle [4] by formally defining the "time-lock public key encryption" (TLPKE) scheme, and demonstrate a construction using a repeated squaring assumption in a group of unknown order (Sec. III).
- We introduce and construct a "tight short-lived signature" scheme from our TLPKE scheme (Sec. IV).
- We conduct a proof-of-concept implementation study and analyze the performance of our construction (Sec. IV).

II. Technical Preliminaries

Basic Notation: Given a set $\mathcal{X}$, we denote by $x \xleftarrow{\$} \mathcal{X}$ the process of sampling a value x from the uniform distribution on $\mathcal{X}$. Supp($\mathcal{X}$) denotes the support of the distribution $\mathcal{X}$. We denote by $\lambda \in \mathbb{N}$ the security parameter. A function `negl`: $\mathbb{N} \to \mathbb{R}$ is negligible if it is asymptotically smaller than any inverse-polynomial function, namely, for every constant $\epsilon > 0$ there exists an integer N_ϵ and for all $\lambda > N_\epsilon$ such that $\mathtt{negl}(\lambda) \leq \lambda^{-\epsilon}$.

Number Theory: We assume that $N = p \cdot q$ is the product of two large secret and *safe* primes and $p \neq q$. We say that N is a strong composite integer if $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes, where p' and q' are also prime. We say that $\mathbb{Z}_N$ consists of all integers in $[N]$ that are relatively prime to N (i.e., $\mathbb{Z}_N = \{x \in \mathbb{Z}_N : \mathtt{gcd}(x, N) = 1\}$).

Repeated Squaring Assumption: The repeated squaring assumption [4] roughly says that there is no parallel algorithm that can perform T squarings modulo an integer N significantly faster than just doing so sequentially, assuming that N cannot be factored efficiently, or in other words `RSW` assumption implies that factoring is hard. More formally, no adversary

can factor an integer $N = p \cdot q$ where p and q are large secrets and "safe" primes (see [2] for details on "safe" primes). Repeated squaring $\texttt{RSW} = (\texttt{Setup}, \texttt{Sample}, \texttt{Eval})$ is defined below. Moreover, we define a trapdoor evaluation $\texttt{RSW.tdEval}$ (which enables *fast* repeated squaring evaluation), from which we can derive an actual output using trapdoor in $\mathsf{poly}(\lambda)$ time.

- $N \leftarrow \texttt{RSW.Setup}(\lambda)$: Output $\mathsf{pp} = (N)$ where $N = p \cdot q$ as the product of two large (λ-bit) randomly chosen secret and safe primes p and q.
- $x \leftarrow \texttt{RSW.Sample}(\mathsf{pp})$: Sample a random instance x.
- $y \leftarrow \texttt{RSW.Eval}(\mathsf{pp}, T, x)$: Output $y = x^{2^T} \mod N$ by computing the T sequential repeated squaring from x.
- $y \leftarrow \texttt{RSW.tdEval}(\mathsf{pp}, \mathsf{sp} = \phi(N), x)$: To compute $y = x^{2^T} \mod N$ *efficiently* using the trapdoor as follows:
  - Compute $v = 2^T \mod \phi(N)$.
    Note: $(2^T \mod \phi(N)) \ll 2^T$ for large T
  - Compute $y = x^v \mod N$.
    Note: $x^{2^T} \equiv x^{(2^T \mod \phi(N))} \equiv x^v \pmod N$.

Assumption 1 (T-Repeated Squaring Assumption without Trapdoor [4])**.** *For every security parameter $\lambda \in \mathbb{N}$, $N \in \mathsf{Supp}(\texttt{RSW.Setup}(\lambda))$, $x \in \mathsf{Supp}(\texttt{RSW.Sample}(N))$, and a time-bound parameter T, computing the $x^{2^T} \mod N$ without knowledge of a trapdoor or secret parameter sp using the $\texttt{RSW.Eval}$ algorithm requires T-sequential time for algorithms with $\mathsf{poly}(\log(T), \lambda)$-parallel processors.*

III. Time-Lock Public Key Encryption

The notion of time-sensitive cryptography was introduced by Rivest, Shamir, and Wagner [4] in 1996, in "Time-lock puzzles and timed-release Crypto" (TLP). They presented a construction using repeated squaring in a finite group of unknown order, resulting in an encryption scheme. This scheme allows the holder of a trapdoor to perform "fast" encryption or decryption, while others without the trapdoor can only do so slowly (requiring T sequential computations).

For the purpose of our tight short-lived signature protocol, we require and define a variation of TLP. We follow the definitions given in [4], altered to fit the public key encryption paradigm, rather than symmetric key encryption. This variation, which we refer to as "Time-Lock Public Key Encryption" (TLPKE)[1], can be described as a "public key encryption scheme with sequential and computationally intensive derived private key generation".

Protocol: The formal details of our TLPKE construction from repeated squaring, is $\texttt{TLPKE} = (\texttt{Setup}, \texttt{Eval}, \texttt{Encrypt}, \texttt{Decrypt})$ specified in Algorithm 1.

Algorithm 1: Time-Lock Public Key Encryption

- $\texttt{TLPKE.Setup}(\lambda, T)$
  1) Call and generate $N \leftarrow \texttt{RSW.Setup}(\lambda)$
  2) Generate an input $x \in \mathbb{Z}_N^* \leftarrow \texttt{RSW.Sample}(\mathsf{pp})$
  3) Generates a key pair (pk, sk) for a semantically secure public-key encryption scheme: $\texttt{PKE} = (\texttt{GenKey}, \texttt{Enc}, \texttt{Dec})$.
  4) Encrypt the sk as $ek = sk + x^{2^T} \mod N$
  5) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ *efficiently* using the trapdoor evaluation $\texttt{RSW.tdEval}(\mathsf{pp}, \mathsf{sp} = \phi(N), x)$.
  6) **return** $\mathsf{pp} = (N, T, x, pk, ek)$.
- $\texttt{TLPKE.Eval}(\mathsf{pp})$
  1) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ using $\texttt{RSW.Eval}(\mathsf{pp}, T, x)$
  2) Extract the decryption key $sk = ek - y$
  3) **return** (y, sk)
- $\texttt{TLPKE.Encrypt}(\mathsf{pp}, M)$
  1) Encrypt a message M with key pk and a standard encryption $\texttt{Enc}$, to obtain the ciphertext $C_M = \texttt{Enc}(pk, M \parallel x)$ and **return** C_M.
- $\texttt{TLPKE.Decrypt}(\mathsf{pp}, sk, y, C_M)$
  1) Decrypt the message as $M \parallel x = \texttt{Dec}(sk, C_M)$
  2) Parse $M \parallel x$ and **return** the message M

[1]TLP with public key encryption instead of symmetric key encryption

IV. Tight Short-Lived Signature

Syntax and Security Definitions: Here, we recall and modify the definition of short-lived signatures (SLS) from [1] and define our tight SLS as follows:

Definition IV.1 (Tight Short-Lived Signatures)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and a space of random beacon $\mathcal{R} \geq 2^\lambda$. A short-lived signature $\texttt{SLS}$ is a tuple of four probabilistic polynomial time algorithms $(\texttt{Setup}, \texttt{Sign}, \texttt{ForgeSign}, \texttt{Verify})$, as follows:*

- $\texttt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk)$*, is randomized algorithm that takes a security parameter λ and outputs public parameters pp and a secret key sk (the sk can **only** be accessed by the $\texttt{SLS.Sign}$ algorithm). The public parameter pp contains an input domain $\mathcal{X}$, an output domain $\mathcal{Y}$, and time-bound parameter T.*
- $\texttt{Sign}(\mathsf{pp}, m, r, sk) \rightarrow \sigma$*, takes a public parameter pp, a secret parameter sp, a message m and a random beacon r, and outputs (in time less than the predefined time bound T) a signature σ.*
- $\texttt{ForgeSign}(\mathsf{pp}, m, r) \rightarrow \sigma$*, takes a public parameter pp, a message m and a random beacon r, and outputs (in time **exactly** T) a signature σ.*
- $\texttt{Verify}(\mathsf{pp}, m, r, \sigma) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$*, is a deterministic algorithm takes a public parameter pp, a message m and a random beacon r and a signature σ, and outputs accept if σ is the correct signature on m and r, otherwise outputs reject.*

A $\texttt{SLS}$ must satisfy the three properties **Correctness** (Definition IV.2), **Existential Unforgeability** (Definition IV.3), and **Indistinguishability** (Definition IV.4) as follows:

Definition IV.2 (Correctness)**.** *A $\texttt{SLS}$ is correct (or complete) if for all $\lambda \in \mathbb{N}$, m, and $r \in \mathcal{R}$ it holds that,*

$$Pr\left[\texttt{Verify}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{matrix}\texttt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \texttt{Sign}(m, r, sk) \rightarrow \sigma\end{matrix}\right] = 1$$

Definition IV.3 (T-Time Existential Unforgeability)**.** *A $\texttt{SLS}$ has T-time existential unforgeability if $\forall\ \lambda, T \in \mathbb{N}$, m and $r \in \mathcal{R}$, and all pairs of PPT algorithms $(\mathcal{A}, \mathcal{A}')$, such an $\mathcal{A}$ (offline) can run in total time $\mathsf{poly}(T, \lambda)$ and in a parallel running time of $\mathcal{A}'$ (online) on at most $\mathsf{poly}(\log T, \lambda)$-processors is less than T, there exists a negligible function $\texttt{negl}$ such that,*

$$Pr\left[\begin{matrix}\texttt{Sign}(m, r, sk) \rightarrow \sigma \\ \texttt{Verify}(\mathsf{pp}, m^*, r, \sigma^*) \\ = \mathsf{accept}\end{matrix} \middle| \begin{matrix}\texttt{Setup}(\lambda) \rightarrow (\mathsf{pp}, sk) \\ \mathcal{A}(\mathsf{pp}, \lambda, T) \rightarrow \alpha \\ \mathcal{A}'(\mathsf{pp}, m^*, r, \alpha) \rightarrow \sigma^*\end{matrix}\right] \leq \texttt{negl}(\lambda)$$

Definition IV.4 (Indistinguishability)**.** *A* `SLS` *is computationally indistinguishable (and statistically indistinguishable; when taken over the random coins used by each algorithm and randomly generated private parameters) if for all* $\lambda \in \mathbb{N}$*,* m*, and* $r \in \mathcal{R}$ *it holds that,*

$$\left| Pr\left[\mathcal{A}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{matrix} \texttt{Setup}(\lambda) \to (\mathsf{pp}, sk) \\ \texttt{Sign}(m, r, sk) \to \sigma \end{matrix}\right] - Pr\left[\mathcal{A}(\mathsf{pp}, m, r, \sigma) = \mathsf{accept} \middle| \begin{matrix} \texttt{Setup}(\lambda) \to (\mathsf{pp}, sk) \\ \texttt{ForgeSign}(\mathsf{pp}, m, r) \to \sigma \end{matrix}\right] \right| \leq \texttt{negl}(\lambda)$$

Our scheme, formalized in Definition IV.1, presents an efficient generalized framework for short-lived signatures (Algorithm 2) that is compatible with all signature schemes. However, note that while our Indistinguishability definition (Definition IV.4) compares distributions of output, some signature schemes are deterministic (e.g., BLS, RSA signature). In such cases, it is necessary for `Sign` and `Forge` to produce the exact signature (e.g., Schnorr signature) with overwhelming probability.

Protocol Design: The formal construction of tight short-lived signatures `SLS` = (`Setup`, `Sign`, `ForgeSign`, `Verify`) using our TLPKE is specified in Algorithm 2.

Algorithm 2: Tight Short-Lived Signatures from TLPKE

- $\texttt{SLS.Setup}(\lambda)$
  1) Call and generate $N \leftarrow \texttt{RSW.Setup}(\lambda)$
  2) Generate an input $x \in \mathbb{Z}_N^* \leftarrow \texttt{RSW.Sample}(\mathsf{pp})$.
  3) Choose a time bound parameter $T \in T(\lambda)$.
  4) Generates a key pair $(pk, sk) = \Pi_{\texttt{KeyGen}}(\lambda)$ for a Signature scheme: $\Pi =$ (`KeyGen`, `Sign`, `Verify`).
  5) Compute $y = x^{2^T} \mod N \in \mathbb{Z}_N^*$ *efficiently* using the trapdoor evaluation $\texttt{RSW.tdEval}(\mathsf{pp}, \mathsf{sp} = \phi(N), x)$.
  6) Encrypts the sk as $ek = sk + x^{2^T} \mod N$
  7) Output public parameter $\mathsf{pp} = (N, T, x, pk, ek)$[a] and secret key sk generated by Π (sk can only be accessed by the $\texttt{SLS.Sign}$).
- $\texttt{SLS.Sign}(m, r, sk)$
  1) Compute $M = \mathcal{H}(m \parallel r)$.
  2) Compute a signature $\sigma = \Pi_{\texttt{Sign}}(sk, M)$.
  3) Output a short-lived signature (σ, r).
- $\texttt{SLS.ForgeSign}(\mathsf{pp}, m, r)$
  1) Compute $M = \mathcal{H}(m \parallel r)$.
  2) Call and extract $(y, sk) = \texttt{TLPKE.Eval}(\mathsf{pp})$.
  3) Compute a forge signature $\sigma = \Pi_{\texttt{Sign}}(sk, M)$[b].
  4) Output a forge short-lived signature (σ, r)
- $\texttt{SLS.Verify}(\mathsf{pp}, m, r, \sigma)$
  1) Compute $M = \mathcal{H}(m \parallel r)$.
  2) Check that $\Pi_{\texttt{Verify}}(pk, M, \sigma)$[c].

[a] The pp can be generate by calling $\texttt{TLPKE.Setup}(\lambda)$.

[b] Forge signature σ can be computed in time not much more than the sequentiality bound ***exactly*** T even on a parallel computer with $\mathsf{poly}(\log T, \lambda)$ processors. Therefore, our SLS is Tight Short-Lived Signature.

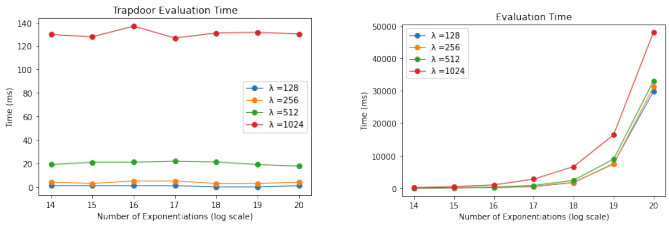
[c] The signature verification algorithm can be computed in $O(1)$ time.

Theorem IV.1 (Tight Short-Lived Signatures)**.** *Assuming that* $\mathcal{H}$ *is a random oracle,* `RSW` *is the repeated point squaring assumption, and* `TLPKE` *is a time-lock public key encryption scheme (see* Algorithm 1*), it holds that the protocol* `SLS` *(*Algorithm 2*) is a tight short-lived signature scheme.*

Proof Sketch. The correctness of the `SLS` scheme is proven by the correctness of the underlying time-lock public key encryption `TLPKE`. Indistinguishability is trivial as the signing and forgery produce the exact signature using the underlying signature scheme, given that the `TLPKE.Eval` (key extraction) and `TLPKE.Decrypt` (decryption) algorithms of the underlying `TLPKE` are deterministic. The T-Time Existential Unforgeability is a direct result of the sequentiality and security (T-IND-CPA Security) property of the underlying `TLPKE` and modeling $\mathcal{H}$ as a random oracle. □

Experimental Results: We use Python to implement `RSW` primitive (and hence our proposed TLPKE and tight SLS). The experiments are performed using a Windows 11 system with Intel(R) Core(TM) i5-1035G1 CPU @1.00GHz with 8 GB RAM. Note that `RSW` is the underlying required primitive of our repeated squaring-based TLPKE and tight SLS. Hence, in this poster, we do not provide detailed simulation results for TLPKE and tight SLS, focusing instead on profiling the underlying workhorse primitive.

In Figure 1, we show the experimental results for `RSW` evaluation. The `RSW.tdEval` (Figure 1a) run time changes linearly with the security parameter λ (the bit length of N is derived from the bit length of λ). As shown in Figure 1b, the time taken to compute the `RSW.Eval` increases with an increase in the number of exponentiations. Changes in time T yield a great variation in the evaluation time.



(a) Trapdoor Evaluation Time. (b) Evaluation Time.

Fig. 1: Trapdoor Evaluation and T-Sequential Evaluation of the Repeated Squaring `RSW`. j is labelled as "Number of Exponentiations", $T = 2^j$.

V. Future Work

We conclude with an open problem: Arun et al. [1] define *reusable forgeability* property in the context of short-lived proofs (see Sec. 4.1 in [1]), which ensure that one slow computation for a random beacon value (say r) enables efficiently forging a proof for any statement (say x) without performing a full additional slow computation. Furthermore, Arun et al. [1] extend reusable forgeability in the context of short-lived signatures and describe a construction (see Sec. 8.3 in [1]). *In the near future, we hope to construct an efficient* ***tight*** *reusable and forgeable short-lived signature scheme.*

References

[1] A. Arun, J. Bonneau, and J. Clark, "Short-lived zero-knowledge proofs and signatures," *IACR Cryptol. ePrint Arch.*, p. 190, 2022. [Online]. Available: https://eprint.iacr.org/2022/190

[2] K. Pietrzak, "Simple verifiable delay functions," *IACR Cryptol. ePrint Arch.*, 2018. [Online]. Available: https://eprint.iacr.org/2018/627

[3] B. Wesolowski, "Efficient verifiable delay functions," 2018, p. 623. [Online]. Available: https://eprint.iacr.org/2018/623

[4] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.