timelock.zone

2 time-locked cryptographic protocols
and a public time-locked cryptographic service
enabling anyone to encrypt data for future decryption

Aragon ZK Research

July 2023 - ***Draft***

Abstract

We describe two protocols that can be used for time-locked encryption under the assumption that a trusted parts publishes a certain type of random beacon at regular intervals. The first protocol, called zk-TLCS, is quite simple but the parties participating in the key-generation process are required to send a ZK proof of the correctness of its parameters. The second protocol, TLCS, does not require a ZK proof, making it much more efficient and generic.

We then present our *timelock.zone* service, which will be based on the TLCS protocol. It will enable anyone to encrypt data for decryption in the future, or to commit data for opening in the future. It has the following key characteristics:

1. Public keys for periods far into the future are always available;
2. Support for several cryptographic schemes, with more to be added in the future;
3. Relies on trusted randomness (drand beacon) published by the League of Entropy;
4. Possibility of public participation;
5. The correctness and security of the scheme is guaranteed as long as a single party participating in the public key computation is honest;
6. These parties do not need to be present when the private key is revealed;
7. No trusted setup.

1 Introduction

Time-locked encryption refers to cryptographic systems which guarantee that ciphertexts will be decipherable only at a certain time in the future, after which even an encoder cannot prevent decryption. Such systems are also variously referred to as time-lapse, time-based, time-dependent, delayed unlocking etc. cryptographic protocols.

In this paper we introduce two generic protocols, zk-TLCS and TLCS, that can be used to implement time-locked encryption for a wide range of public key encryption schemes. The only assumption is the existence of a certain type of trusted Beacon. The master public key of the chosen protocol is generated in a distributed way among a set of parties. The security of the scheme is guaranteed as long as there exist at least a single honest party that participates in the key generation protocol.

We also present the outline of a future public service built on the TLCS protocol, called *timelock.zone*. It will use the *drand* beacon published by the League of Entropy (cf Section 2.3) and will support the most common cryptographic schemes. This service will enable anyone to encrypt data for decryption at precise times in the future: hourly for the next two years and daily for the next ten years.

Time-lock cryptography has many applications. One of them is in e-voting systems: ballots are encrypted during the voting period, and can be publicly decrypted after the voting period is over.

This allows for transparent and independent verification of results. Indeed, the protocols described in this paper are the result of our work on a voting system for which we needed a time-lock service.

2 Random Beacon

We assume that there exists a trusted party that generates and publishes a random Beacon (hereafter simply called Beacon) at regular intervals.

2.1 Beacon Parameters

The private parameter of the Beacon is its private key $\mathbf{sk}_L \in \mathbb{Z}_p$. Its public parameters are:

$$\mathsf{pp}_L = \langle p, \mathbb{G}_1, g_1, \mathbb{G}_2, g_2, \mathbb{G}_T, \mathbf{e}, H, \mathsf{PK}_L \rangle$$

- $\mathbb{G}_1$, $\mathbb{G}_2$ and G_T are groups of prime order p.
- $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by g_1 and g_2 respectively.
- $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate pairing function.
- $H : \mathbb{Z}_p \to \mathbb{G}_1$ is a "number to point" hash function.
- $\mathsf{PK}_L = g_2^{\mathbf{sk}_L}$ is the public key.

Note that in case of a symmetric pairing we will have $\mathbb{G}_1 = \mathbb{G}_2$ and $g_1 = g_2$.

2.2 Beacon Output

At regular time intervals, the Beacon publishes the signature $H(R)^{\mathbf{sk}_L}$. R is called a "round" and corresponds to a UNIX epoch at which the value is published. The publication time is known in advance.

2.3 League of Entropy

In our case, this trusted party will be the League of Entropy (LoE) [11] which is a collaborative project to provide a verifiable, decentralized randomness beacon called drand [7]. Its founding members are Cloudflare, École Polytechnique Fédérale de Lausanne, Kudelski Security, Protocol Labs and the University of Chile. As of June 2023, the number of participants has risen to 18.

LoE uses the BLS12-381 curve for signing, with an asymmetric pairing function. It started generating its "pedersen-bls-chained" beacon in 2019 and declared it production-ready in August 2020. This beacon produces a random value every 30 seconds and is not suitable for time-based encryption. In March 2023, LoE launched its "bls-unchained-on-g1" beacon, which is generated every 3 seconds and is suitable for time-based encryption.

The LoE beacons are secure as long as a majority of LoE members do not collude.

3 Protocols for a Cryptographic Timelock Service

Our goal is to define a protocol in which a set of participants cooperate in computing a public key of some encryption scheme Π^{enc}, so that the corresponding private key can be obtained at some future time, but not before. The participants have access to the data from the Beacon.

The scheme Π^{enc} is defined over a group $\mathbb{G} = \langle g \rangle$ of prime order q, and we assume that there exists a "point to number" hash function $\mathscr{H} : \mathbb{G}_T \to \mathbb{Z}_q$ that acts as a random oracle.

The two protocols we will present have four non-overlapping phases for each round R. During the *contribution phase*, participants submit their public parameters. During the *public key publication phase*, the master public key $\mathsf{MPK}(R)$ is computed. This is followed by the *waiting phase*, during which where $\mathsf{MPK}(R)$ is used by third parties to produce ciphertexts. In the fourth and final phase, the *decryption phase*, begins after the Beacon has published the signature $H(R)^{\mathbf{sk}_L}$. The secret key $\mathbf{sk}(R)$ is computed, and any data encrypted using $\mathsf{MPK}(R)$ can be decrypted.

Note that we generally write private variables in bold and public variables using a regular font. Field elements are written in lowercase and group elements in uppercase, with the exception of the generators g, g_1 and g_2.

3.1 Protocol 1: zk-TLCS

The first protocol, which we call zk-TLCS, is quite simple but requires a zero-knowledge (ZK) proof.

3.1.1 Participant contribution

Every participant follows the protocol as shown in Table 1.

	public	private
1		$\mathbf{sk} \xleftarrow{\$} \mathbb{Z}_q$
2	$\mathsf{PK} \leftarrow g^{\mathbf{sk}}$	
3		$\mathbf{t} \xleftarrow{\$} \mathbb{Z}_p$
4	$T \leftarrow g_2^{\mathbf{t}}$	
5		$\mathbf{Z} \leftarrow \mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}}$
6	$y \leftarrow \mathscr{H}(\mathbf{Z}) \oplus \mathbf{sk}$	
7	proof π	

Table 1: Protocol zk-TLCS

Note that in line 6, $\mathbf{sk}$ must be represented as a bitstring using a suitable serialisation method. As to π, it is a ZK proof for the following inputs and assertions:

- generic public inputs: $\mathsf{PK}_L, H(R)$
- participant-specific public inputs: PK, T, y.
- private inputs: $\mathbf{sk}, \mathbf{t}$
- assert $PK = g^{\mathbf{sk}}$
- assert $T = g_2^{\mathbf{t}}$
- assert $y = \mathscr{H}(\mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}}) \oplus \mathbf{sk}$

The contribution of each participant are the public parameters $\mathsf{pp} = (\mathsf{PK}, T, y, \pi)$. After computing these public parameters, the private values $\mathbf{sk}, \mathbf{t}$ and $\mathbf{Z}$ must be securely discarded, as they constitute the so-called "toxic waste" of the protocol. Note that in this protocol, as single honest participant will ensure that the scheme is secure.

In the following, we will denote the public parameters of participant i by $\mathsf{pp}^{(i)} = (\mathsf{PK}^{(i)}, T^{(i)}, y^{(i)}, \pi^{(i)})$ and the private parameters by $\mathbf{sk}^{(i)}, \mathbf{t}^{(i)}, \mathbf{Z}^{(i)}$.

3.1.2 Public key computation

Once all participants have published their contributions, the master public key will simply be the product of the individual public keys:

$$\mathsf{MPK}(R) = \prod_i \mathsf{PK}^{(i)} \tag{1}$$

This master public key can then be used for encryption, commitments and signatures with the chosen cryptographic scheme.

3.1.3 Private key computation

At round R, the Beacon publishes the value $H(R)^{\mathbf{sk}_L}$. For every participant i we can compute:

$$\begin{aligned}\mathbf{e}(H(R)^{\mathbf{sk}_L}, T^{(i)}) &= \mathbf{e}(H(R)^{\mathbf{sk}_L}, g_2^{\mathbf{t}^{(i)}}) \\ &= \mathbf{e}(H(R), g_2^{\mathbf{sk}_L})^{\mathbf{t}^{(i)}} \\ &= \mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}^{(i)}} \\ &= \mathbf{Z}^{(i)}\end{aligned}$$

We then obtain $\mathbf{sk}^{(i)}$ via:

$$\mathbf{sk}^{(i)} = \mathscr{H}(\mathbf{Z}^{(i)}) \oplus y^{(i)}$$

Once this is done for all participants, the private key for round R is:

$$\mathbf{sk}(R) = \sum_i \mathbf{sk}^{(i)}$$

3.2 Protocol 2: TLCS protocol

The second protocol, which we call TLCS, is seemingly more complex but avoids general-purpose NIZK proofs. It is this protocol that will be used by the timelock service. The integer k is the security parameter, and we define $\mathcal{H}_k$ as the first k bits of a cryptographic hash function $\mathcal{H}$.

3.2.1 Participant contribution

The non-interactive algorithm for participating in the key generation for round R is shown in Table 2. As before, for simplicity the index of the party is omitted in that table. We define:

$$\overrightarrow{\mathsf{PK}_0} = (\mathsf{PK}(1,0), \mathsf{PK}(2,0), ..., \mathsf{PK}(k,0))$$

The vectors $\overrightarrow{\mathsf{PK}_1}$, $\overrightarrow{T_0}$, etc. are defined in a similar way. The output of party i is then:

$$\mathsf{pp}^{(i)} = \left(\mathsf{PK}^{(i)}, \overrightarrow{\mathsf{PK}_0}^{(i)}, \overrightarrow{\mathsf{PK}_1}^{(i)}, \overrightarrow{T_0}^{(i)}, \overrightarrow{T_1}^{(i)}, \overrightarrow{y_0}^{(i)}, \overrightarrow{y_1}^{(i)}, \overrightarrow{t^*}^{(i)}\right)$$

	public	private
1		$\mathbf{sk} \xleftarrow{\$} \mathbb{Z}_q$
2	$\mathsf{PK} \leftarrow g^{\mathbf{sk}}$	
		$\forall j \in [k]$
3		$\mathbf{sk}(j,0) \xleftarrow{\$} \mathbb{Z}_q$
4		$\mathbf{sk}(j,1) \leftarrow \mathbf{sk} - \mathbf{sk}(j,0)$
	$\forall j \in [k]$	
5	$PK(j,0) \leftarrow g^{\mathbf{sk}(j,0)}$	
6	$PK(j,1) \leftarrow g^{\mathbf{sk}(j,0)}$	
		$\forall j \in [k]$
7		$t(j,0) \xleftarrow{\$} \mathbb{Z}_p$
8		$t(j,1) \xleftarrow{\$} \mathbb{Z}_p$
	$\forall j \in [k]$	
9	$T(j,0) \leftarrow g_2^{\mathbf{t}(j,0)}$	
10	$T(j,1) \leftarrow g_2^{\mathbf{t}(j,1)}$	
		$\forall j \in [k]$
11		$\mathbf{Z}(j,0) \leftarrow \mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}(j,0)}$
12		$\mathbf{Z}(j,1) \leftarrow \mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}(j,1)}$
	$\forall j \in [k]$	
13	$y(j,0) \leftarrow \mathscr{H}(\mathbf{Z}(j,0) \oplus \mathbf{sk}(j,0)$	
14	$y(j,1) \leftarrow \mathscr{H}(\mathbf{Z}(j,1) \oplus \mathbf{sk}(j,1)$	
15	$\vec{b} \leftarrow \mathcal{H}_k(\mathsf{PK}, \overrightarrow{\mathsf{PK}}_0, \overrightarrow{\mathsf{PK}}_1, \overrightarrow{T}_0, \overrightarrow{T}_1, \vec{y}_0, \vec{y}_1)$	
	$\forall j \in [k]$	
16	$t^*(j) \leftarrow \mathbf{t}(j, b_j)$	

Table 2: TLCS Protocol

3.2.2 Interactive protocol

In the interactive version, the participant first executes steps 1 through 14 and sends

$$(\mathsf{PK}, \overrightarrow{\mathsf{PK}}_0, \overrightarrow{\mathsf{PK}}_1, \overrightarrow{T}_0, \overrightarrow{T}_1, \vec{y}_0, \vec{y}_1)$$

to the verifier. The verifier then sends a random bitstring $\vec{b}$ of length k to the participant, who returns $\overrightarrow{t^*}$.

3.2.3 Validity check

The algorithm for checking the validity of data submitted by a participant is shown in Figure 3.2.3.

Figure 1: Algorithm to verify the validity of public parameters sent by Participants

Require: pp is well-formed
1: **for** $j = 1, \ldots, k$ **do**
2: **assert** $PK(j,0) \cdot PK(j,1) == \mathsf{PK}$
3: **assert** $T(j, b_j) == g_2^{t^*(j)}$
4: $\hat{Z}(j) \leftarrow \mathbf{e}(H(R), PK_L)^{t^*(j)}$
5: $\hat{s}(j) \leftarrow \mathscr{H}(\hat{Z}(j)) \oplus y(j, b_j)$
6: **assert** $g^{\hat{s}(j)} == \mathsf{PK}(j, b_j)$
7: **end for**
8: **Output** accept

We start by checking that pp is well-formed, i.e. it has the required number of group/field elements.

In line 2, we check that all the public keys are compatible. This also means that the participant must know the values $\mathbf{sk}(j,0)$ and $\mathbf{sk}(j,1)$, except with negligible probability.

Now remember that if the participant generated pp correctly, then $t^*(j) = \mathbf{t}(j,b_j)$. If that is the case we will have $g_2^{t^*(j)} = g_2^{\mathbf{t}(j,b_j)} = T(j,b_j)$ and the assert in line 3 will pass. We will also have $\hat{Z}(j) = \mathbf{Z}(j,b_j)$ and $\hat{s}(j) = \mathbf{sk}(j,b_j)$, and as a result the assert on line 6 will pass.

3.2.4 Public key computation

Once all participants have published their contributions, the master public key is computed in the same way as in protocol 1:

$$\mathsf{MPK}(R) = \prod_i \mathsf{PK}^{(i)} \tag{2}$$

3.2.5 Private key computation

At round R, the Beacon publishes the value $H(R)^{\mathbf{sk}_L}$. For every participant we can now compute:

$$\begin{aligned}
\mathbf{e}(H(R)^{\mathbf{sk}_L}, T(j,0)) &= \mathbf{e}(H(R)^{\mathbf{sk}_L}, g_2^{\mathbf{t}(j,0)}) \\
&= \mathbf{e}(H(R), g_2^{\mathbf{sk}_L})^{\mathbf{t}(j,0)} \\
&= \mathbf{e}(H(R), \mathsf{SK}_L)^{\mathbf{t}(j,0)} \\
&= \mathbf{Z}(j,0)
\end{aligned}$$

We then obtain $\mathbf{sk}(j,0)$ via:

$$\mathbf{sk}(j,0) = \mathscr{H}(\mathbf{Z}(j,0)) \oplus y(j,0)$$

The values of $\mathbf{Z}(j,1)$ and $\mathbf{sk}(j,1)$ are computed similarly, and we therefore obtain $\mathbf{sk}(j) = \mathbf{sk}(j,0) + \mathbf{sk}(j,1)$ which should be equal to $\mathbf{sk}$ for all $j \in [k]$ if the participant was honest.

Once this is done for all participants, the private key for round R is then again:

$$\mathbf{sk}(R) = \sum_i \mathbf{sk}^{(i)}$$

3.2.6 Soundness error

The soundness error will be 2^{-k} so there is a trade-off: we have freedom to choose k as large as possible to reduce the soundness error and as small as possible to make the system more efficient. Any polynomial dishonest prover can cheat with prob 2^{-k}. This is true only for the interactive version.

When done non-interactively, the soundness is no longer statistical because the prover can make brute force attacks. If TLCS keys are supposed to be used for short periods we consider that a security parameter between 80 and 100 should be sufficient for most applications. Observe that the error also scales down with the number of parties participating in the protocol. So concretely, we suggest a security parameter of $k = 80$ for a small number of participants (e.g. less than 10) and keys with periods of validity of a few months and $k = 100$ for hundreds of participants and keys with a validity of several years.

3.3 On extending the protocol to generic one-way functions

In the two protocols presented above, we have assumed the existence of a group $\mathbb{G} = \langle g \rangle$ of order q as the basis of our cryptographic scheme. We can generalise these protocols to generic one-way functions (OWF) as long as these have a suitable homomorphic property. All groups have this homomorphic property, as $g^x g^y = g^{x+y}$, but it also holds for RSA and for certain post-quantum OWFs. We leave as future work whether and how to extend our protocols to RSA and post-quantum encryption schemes.

	public	private
1		$\mathbf{sk} \xleftarrow{\$} A$
2	$\mathsf{PK} \leftarrow f(\mathbf{sk})$	
3		$\mathbf{t} \xleftarrow{\$} \mathbb{Z}_p$
4	$T \leftarrow g_2$	
5		$Z \leftarrow \mathbf{e}(H(R), \mathsf{PK}_L)^{\mathbf{t}}$
6	$y \leftarrow \mathscr{H}(Z) \oplus \mathbf{sk}$	
7	proof π	

Table 3: zk-TLCS protocol for OWF

Let $f : (A, *) \to (B, \circ)$ be such a one-way function, with the property $f(x) \circ f(y) = f(x * y)$. We can then modify the zk-TLCS Protocol as shown in Table 3. The only requirement is that elements of A can be represented as bitstrings of a suitable length. If n denotes the number of participants, the master public key is then obtained as:

$$\mathsf{MPK}(R) = \mathsf{PK}^{(1)} \circ ... \circ \mathsf{PK}^{(n)} = f(sk^{(1)} * ... * sk^{(n)})$$

Once $H(R)^{\mathbf{sk}_L}$ is published, we can recover $\mathbf{sk}^{(i)}$ for all participants as before, because it is represented as a bitstring. The private key for round R is then:

$$\mathbf{sk}(R) = \mathbf{sk}^{(1)} * ... * \mathbf{sk}^{(n)}$$

The TLCS protocol can be adapted in a similar way.

Additional Note. In order to make the protocol quantum secure, we could replace the cryptosystem with homomorphic post-quantum ones such as schemes proposed in [10] and [1]. In this stage, it is yet to be ascertained whether these schemes are compatible with the TLCS protocol, warranting further investigation in future research.

3.4 Security Analysis

A timelock protocol should satisfy two main security properties. The first one, which we call *soundness*, should guarantee that if a contribution is accepted and published on the blockchain during an execution of the protocol for a round R then the master public key $\mathsf{MPK}(R)$ obtained by aggregating all contributions is such that at round R, $\mathsf{MPK}(R)$ can be inverted to obtain $\mathbf{sk}(R)$ using as auxiliary information the secret key SK_L published by the beacon. A protocol that is not sound would be one in which a malicious adversary is able to inject a contribution that makes $\mathsf{MPK}(R)$ not invertible at round R. Our main protocol satisfies soundness since the interactive version of our protocol is indeed a sigma protocol and as such is knowledge sound.

The other main security property that a TLCS protocol should satisfy is what we call *privacy*. We have seen that the algorithm requires the participants to delete the private variables $\mathbf{sk}$, $\mathbf{t}$ and $\mathbf{Z}$ from memory at the end of the computation. If only a single participants acts honestly and deletes these private variables, then the secret key $\mathbf{sk}$ corresponding to $\mathsf{MPK}(R)$ is protected until

round R, assuming that the majority of LoE members are honest. This means that the only added assumption beyond LoE is that there is a single honest party. Observe that, if participation in the protocol were to be opened to the public, then if someone does not trust the system they can decide to participate in the protocol themselves.

We leave as future work a formal security model for TLCS protocols and a security proof for our main protocol.

4 Description of the timelock.zone cryptographic service

4.1 Overview

4.1.1 User interface

There will be a user interface at `https://timelock.zone` to query any keys published by the Timelock service.

4.1.2 Timelock periods

For every supported scheme, we plan to make available public keys for times in the future as follows: every hour for the next 2 years, and every day for the next 10 years [1].

4.1.3 Supported schemes

We plan to support a wide variety of schemes to cover use cases not only in web3, but also in more traditional settings. Among the curves supported are Curve 25519, P-256, P-384, P-521, secp256k1, MNT6 [2], the G_1 group of BLS12-381, and BabyJubJub [14]. It is also easy to support secure multiplicative groups at the cost of larger data volumes.

4.1.4 The timelock.zone blockchain

The timelock.zone blockchain will be used to store the shares of the various public keys, and also to ensure that consensus is reached as to which shares are used to compute each public key. It is being built using Gears [9], a Rust implementation of the Cosmos SDK [5].

4.2 Lifecycle of a public/private keypair

This outlines the different phases of the process for a specific keypair.

Contribution phase To generate a private key, we need several parties to generate "shares" of the private key. These "contributors" make their shares publicly available on the timelock blockchain. At least one of these parties needs to be honest, meaning it discards the private data generated during the computation of the share. Note that the software for generating shares will be open source, and it can also be used to verify if shares are valid.

Public key publication The validators are the parties who will compute the public key. This group can be different from the group of contributors, but in practice we expect that most validators will also be contributors. Once the contribution phase is over, the role of the validators is decide which shares to include in the public key generation process. Obviously, only valid shares should be included. Once the set of shares is decided, the public key can be generated from these shares and published on the blockchain.

Waiting period During this phase, the private key corresponding to the previously published public key is not yet available. The shares corresponding to the public key are visible on the blockchain and can be used to verify the validity of the public key.

Private key publication Once the corresponding drand signature has been published, the private key can be computed by anyone as long as all shares are available on the blockchain. The original contributors do not need to be involved. The validators jointly publish the private key on the blockchain. Once this is done, the shares are no longer needed, although they will remain available on archival nodes.

[1] These parameters are still subject to change

4.3 Key generation

A contribution or share is represented by the tuple $(R, \mathsf{scheme}, \mathsf{pp})$, where:

- R is a round, which corresponds to a UNIX epoch
- scheme is the identifier of the scheme
- pp (which stands for "public parameters") is a dataset that can be used to compute a public key, by aggregating it with other datasets

The contribution does not include information about who submitted the contribution, but that information will be available on the blockchain. A contribution is valid if:

- it has been submitted by a validating node, or by a whitelisted participant (conditions for whitelisting will be determined in the future)
- the scheme is supported
- it has been submitted during the contribution period for R
- verification of data is successful

A timelock dataset is represented by $(R, \mathsf{scheme}, \overrightarrow{\mathsf{pp}}, \mathsf{MPK}, \mathbf{sk})$:

- $\overrightarrow{\mathsf{pp}}$ is a vector of participant contributions submitted during the submission period
- MPK is the master public key, computed using $\overrightarrow{\mathsf{pp}}$ once the contribution period ends
- **sk** is the secret key, computed using $\overrightarrow{\mathsf{pp}}$ and $sig_L(R)$ once the latter become available

A timelock dataset gets initialised when the first valid contribution $(R, \mathsf{scheme}, \mathsf{pp})$ for a given round R and scheme is received.

4.4 Contribution periods

As stated above, we aim to publish public keys as follows: every hour for 2 years, and every day for 10 years, with a contribution period of 14 days. For a given date D and full hour there is a well-defined contribution period:

- For D:12 (noon) the contribution period will be (D - 10y - 14d):12 to (D - 10y):12
- For D:hh (hh not 12) the contribution period will be (D - 2y - 14d):hh to (D - 2y):hh

This means that

- at D:12 (noon of day D):
  - start contribution period for (D + 10y + 14d):12
  - end contribution period for (D + 10y):12
- at D:hh (top of the hour hh of day D, with hh not 12):
  - start contribution period for (D + 2y + 14d):hh
  - end contribution period for (D + 2y):hh

Obviously we will need to compute numerous public keys when launching the service.

Contributions will be accepted if and only if the block timestamp of the last block is within the contribution period. Public keys for a round R (and a given scheme) are computed as soon as there is a block whose timestamp is after the end of the contribution period. Finally, the private keys are computed as soon as the drand signature for round R is published.

4.5 Expected data volumes

We estimate that a single share will be roughly 50 kB, that is 0.05 MB [2]. If we want to generate keys on an hourly basis for one year in advance, with 10 contributors and for 10 schemes, this will represent approximately 365*24*10*10*0.05 kB = 43,800 MB = 43.8 GB of data. This number will grow in proportion to the number of supported schemes, the number of contributors and the number of years for which keys will be generated in advance. To have keys for 20 schemes 2 years in advance with 20 contributors, we will need to store 350 GB of data. To this, we need to add shares of daily keys for periods exceeding 2 years. Clearly, such data volumes cannot be stored on a public blockchain.

5 Related work

The notion of timelock encryption (also referred as timed-release encryption in the literature) was first introduced in the seminal work of Rivest *et al.* [13]. Boneh and Naor [3] later improved on that work and presented a protocol for timed commitments, which allows a sender to commit to a message without revealing its content for a specified duration.

Liu *et al.* [12] put forth a protocol for sending messages to the future based on blockchains. Unfortunately, their protocol is based on expensive primitives like extractable witness encryption. Campanelli *et al.* [4] propose a timelock encryption protocol in which a committee of members needs to interact together to transmit messages to future leaders of a PoS ("proof of stake") competition.

The tlock encryption schemes of Gailly *et al.* [8] and the McFly scheme of [6] allow to encrypt to the future in a specific bilinear group setting and as such are not usable as timelock cryptographic services that offer public/secret keys of generic elliptic curve and multiplicative groups. The scheme of Gailly *et al.* is based on the drand service, like TLCS.

References

[1] Khin Mi Mi Aung, Hyung Tae Lee, Benjamin Hong Meng Tan, and Huaxiong Wang. Fully homomorphic encryption over the integers for non-binary plaintexts without the sparse subset sum problem. *Theor. Comput. Sci.*, 771:49–70, 2019.

[2] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. *Algorithmica*, 79(4):1102–1160, 2017.

[3] Dan Boneh and Moni Naor. Timed commitments. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, page 236–254, Berlin, Heidelberg, 2000. Springer-Verlag.

[4] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 151–180. Springer, 2022.

[5] Cosmos sdk. `https://github.com/cosmos/cosmos-sdk`.

[6] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: Verifiable encryption to the future made practical. *IACR Cryptol. ePrint Arch.*, page 433, 2022.

[7] drand beacon. `https://drand.love/`.

[8] Nicolas Gailly, Kelsey Melissaris, and Yolan Romailler. tlock: Practical timelock encryption from threshold BLS. *IACR Cryptol. ePrint Arch.*, page 189, 2023.

[2] The exact size will depend on the scheme, the compression algorithm used, if any, and the serialization.

[9] Gears. `https://github.com/rumos-io/gears`.

[10] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2013.

[11] League of entropy. `https://www.cloudflare.com/leagueofentropy/`.

[12] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Des. Codes Cryptogr.*, 86(11):2549–2586, 2018.

[13] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. *MIT Technical Report MIT-LCS-TR-684*, 1996.

[14] Barry WhiteHat, Marta Bellés, and Jordi Baylina. Erc-2494: Baby jubjub elliptic curve. `https://eips.ethereum.org/EIPS/eip-2494`.