i-TiRE: Incremental Timed-Release Encryption
or
How to use Timed-Release Encryption on Blockchains?

Leemon Baird
Swirlds Labs
leemon@swirldslabs.com

Pratyay Mukherjee
Swirlds Labs
pratyay.mukherjee@swirldslabs.com

Rohit Sinha
Swirlds Labs
rohit.sinha@swirldslabs.com

ABSTRACT

Timed-release encryption can encrypt a message to a future time such that it can only be decrypted after that time. Potential applications include sealed bid auctions, scheduled confidential transactions, and digital time capsules. To enable such applications as decentralized smart contracts, we explore how to use timed-release encryption on blockchains.

Practical constructions in literature rely on a trusted server (or servers in a threshold setting), which periodically publishes an epoch-specific decryption key based on a long-term secret. Their main idea is to model time periods or epochs as *identities* in an *identity-based encryption* scheme. However, these schemes suffer from a fatal flaw: an epoch's key does not let us decrypt ciphertexts locked to prior epochs. Paterson and Quaglia [SCN'10] address this concern by having encryption specify a range of epochs when decryption is allowed. However, we are left with an efficiency concern: in each epoch, the server(s) must publish (via a smart contract transaction) a decryption key of size logarithmic in the lifetime (total number of epochs). For instance, on Ethereum, for a modest lifetime spanning 2 years of 1-minute long epochs, a server must spend over \$6 in gas fees, every minute; this cost multiplies with the number of servers in a threshold setting.

We propose a novel timed-release encryption scheme, where a decryption key, while logarithmic in size, allows *incremental updates*, wherein a short update key (single group element) is sufficient to compute the successive decryption key; our decryption key lets the client decrypt ciphertexts locked to any prior epoch. This leads to significant reduction is gas fees, for instance, only \$0.30 in the above setting. Moreover, ciphertexts are also compact (logarithmic in the total lifetime), and encryption and decryption are on the order of few milliseconds. Furthermore, we decentralize the trust among a number of servers, so as to tolerate up to a threshold number of (malicious) corruptions.

Our construction is based on bilinear pairing, and adapts ideas from Canetti et al.'s binary tree encryption [Eurocypt 2003] and Naor et al.'s distributed pseudorandom functions [Eurocrypt 1999].

1 INTRODUCTION

Timed-release encryption [28] can encrypt a message "locked" to a future time such that a receiver can only decrypt the ciphertext after that time. It opens the door for several novel applications [33, 34]. Examples include: 1) sealed-bid auctions; 2) scheduled confidential transactions (e.g. insider trades); 3) digital equivalent of time capsules[1]; 4) cryptocurrency wallet backups (e.g., escrow or a set of users assisting with key recovery after a deadline).

The existing constructions of timed-release encryption can be broadly divided into two categories: computational reference clocks and trusted time servers. Schemes based on computational reference clocks [10, 11, 15, 19, 34] require the recepient to perform an expensive sequential computation (also called time-lock puzzle) to recover the message. This has the benefit of non-interactive decryption, but is highly inefficient for most applications. The practical alternative is to rely on trusted time server(s) [12, 16, 17, 32, 33] who holds a master secret key, and periodically releases decryption keys at each time epoch (for example, every minute or even every second).

We begin our work with the following question: *how can timed-release encryption be used by smart contracts on blockchains?* Such a primitive would enable privacy-enhancing alternatives to a large ecosystem of decentralized financial applications, such as auctions and decentralized exchanges (DEXs) – in particular, sealed bid auctions and scheduled confidential transactions can rely on the users' orders being secret (until a deadline) while also binding or actionable without requiring any further interaction from the user (e.g., opening in a commit-reveal scheme, which the user will only do if the outcome is favorable)[2]. We study timed-release encryption in the following blockchain-based setting: a set of servers, of which a threshold fraction is assumed to operate correctly, periodically publish (shares of) the decryption key via transactions to an special *aggregator* contract, who then aggregates them for use by other smart contracts, such as for sealed-bid auctions[3].

A long line of constructions [12, 16, 33] operate by deriving an independent key for each epoch by essentially making black-box use of identity-based encryption (where epochs are mapped to identities). While the keys are sufficiently compact (single group element), they suffer from a fatal problem: an epoch's key does not let us decrypt ciphertexts locked to prior epochs, which is a common situation given the nature of our applications – a workaround is to store historical keys, but that incurs on-chain storage that is linear in the lifetime of the system.

Paterson and Quaglia [32] proposed a natural extension, called *time-specific* encryption, wherein the encryption procedure can

[1] A 19^{th} century application: Mark Twain stipulated that his autobiography not be published for 100 years after his death [1].
[2] In that light, timed-release decryption can be viewed in the same light as threshold decryption, but with the added crucial property that a single key can be used across arbitrary many applications and users.
[3] This model is based on blockchain oracles, who provide off-chain data and services, by periodically issuing transactions in exchange for a token payment.

lock the ciphertext to a range of time epochs when decryption is allowed, with only logarithmic increase in the ciphertext size (and logarithmic size keys) – this allows prior decryption with only logarithmic size on-chain storage. However, an efficiency concern remains. Note that each server must publish the latest decryption key on each epoch, which incurs a smart contract transaction with logarithmic size input – the natural design is to use an "incremental" mode of operation where a server publishes only the delta between two consecutive keys; even then, a server would publish $log(T)/2$ new group elements on average in each epoch (where T is the total number of epochs in the system's lifetime).

Input bytes are expensive for on-chain transactions. Consider an example deployment on Ethereum; with a lifetime of $T = 2^{20}$ epochs (roughly 2 years lifetime spanning 1-minute epochs), 100 servers would spend a total of $614 in gas costs for each epoch[4].

For timed-release encryption to be practical on blockchains, incremental updates must be short. That is the objective of this work.

1.1 Our work

We provide a novel solution for timed-release encryption, which has an asymptotic and concrete reduction in on-chain costs. In particular, our scheme has a special *incrementality* property that ensures that the server needs to publish exactly one group element (called the update key) in each epoch, for augmenting the decryption key from epoch τ to $\tau + 1$; this updated decryption key lets decryption of ciphertexts locked to any epoch $\leq \tau + 1$. We call the The decryption key is still logarithmic size, as in [32], so on-chain storage is logarithmic size. Since update keys, and therefore their smart contract transactions, are now constant size, our total cost for 100 servers goes down to $30.7 for each epoch, in the example above. We maintain other characteristics from [32]; our ciphertexts are logarithmic in size, and encryption and decryption operations are also logarithmic time (on the order of few milliseconds).

While our base scheme has a single trusted key-server, we also propose how to further decentralize the system by using threshold cryptography techniques. In particular, we show how instead of a single time-server, we can use n servers, each of them holding only a share of the master secret key, such that any t (for $1 \leq t \leq n$) of them need to publish a share of the update key in any epoch. Furthermore, our threshold scheme is resilient to (up to) $t - 1$ malicious corruptions.

We provide a simple and efficient construction that achieves chosen-plaintext security using an (asymmetric) bilinear pairing on a Gap Diffie-Hellman group – our security reduces to the decision Biliniear Diffie-Hellman (DBDH) assumption in the random oracle model. Our (t out of n) threshold solution is obtained using the key-homomorphic property of our construction similar to threshold BLS signatures [5] or NPR distributed PRFs [29]. Malicious security against $\leq t$ corruptions is achieved using techniques similar to DiSE [2]. The overall solution is still quite efficient. We also show how to obtain CCA-security using a standard (namely Fujisaki-Okamoto [22]) transformation. We emphasize that these two augmentations are done independently such that it is possible to *combine* these properties (CCA and threshold-malicious security together) in any desired way.

Consider a few metrics for our maliciously secure threshold scheme (CCA security), for a sample data point: lifetime of 2^{30} epochs, or roughly 34 years with 1-second epochs. Our key size is logarithmic in the number of epochs; it averages 2.4 KB, depending on the specific epoch. Computing the update key incurs logarithmic number of group operations on the server (2-4 ms on average). The update key is one group element (48 bytes) in each epoch; in the threshold setting, servers publish one group element each. Ciphertexts are also logarithmic in size (0.19-2 KB of ciphertext expansion) and decryption incurs logarithmic number of group operations (35-50 ms).

1.2 Summary of Our Contribution

– We formalize incremental timed-release encryption, called i-TiRE, and in particular its incrementality property (and its extension to the threshold setting).
– We put forward a new efficient construction satisfying our incrementality requirement.
– We provide an open-source implementation and evaluation measuring the sizes of keys and ciphertexts, and the running time of the various algorithms in our threshold i-TiRE scheme.

2 RELATED WORK

Computational Reference Clocks. Instead of having an absolute decryption time, schemes based on time-lock puzzles require the recepient to perform an expensive sequential computation to recover the message, thus imposing a coarse-grained release time. Rivest et al. [34] provide a construction based on repeated squaring modulo a product of two primes. Mahmoody et al. [27] constructs time-lock puzzles in the random oracle model. Liu et al. [26] construct time-locked encryption using a *computational reference clock* (based on Bitcoin hashchains) and an extractable witness encryption scheme, which is not practical.

Trusted Time Servers. Blake and Chan [12] and Cheon et al. [16] provide schemes that are adaptations of the Boneh-Franklin IBE scheme [4]. An similar technique [30] is adapted by Shutter Network to prevent front-running attacks. Neither schemes enable *compact* decryption of prior ciphertexts; in other words, to enable decryption of prior ciphertext one needs to store *all* keys released so far. The scheme of Rabin and Thorpe [33] requires the servers to compute a separate public key for each epoch, whose private component is released during that epoch. This requires the servers to apriori publish a long list of future public keys.

Time-specific Encryption [32]. Perhaps the most relevant to ours is the work by Paterson and Quaglia [32], who introduced a related notion called *time-specific encryption*, where ciphertexts are locked to a range of timestamps. Certainly, our notion of timed-release encryption is a special case of their notion, because one may just fix the upper range to the maximum value of time to obtain a timed-release encryption. We do not focus on achieving time-specific encryption. Instead, we focus on achieving the incrementality property which was not considered before. Taking a closer look at their work [32], we observe that while it is possible to adapt their scheme to our

[4]At the time of writing, as per [38], each byte of a transaction costs 16 gas units; with an average cost of 200 GWei per gas unit, we incur 0.0000032 ETH or roughly $0.01 at $3000 / ETH. So, a single group element of 32 bytes costs roughly $0.30.

setting, it will still fail to achieve the incrementality. On average the update keys in the adapted scheme would consist of $\log(T)/2$ group elements, whereas for our scheme it is always a single group element (recall that the incrementality requires this to be constant size, always). Intuitively, this is due to the fact that they put the path information (root to node) into their keys and use a minimal set cover for the ciphertexts, whereas our ciphertexts have the path information and our decryption keys are corresponding to a minimal set cover via a post-order traversal. Since ciphertexts can not be augmented from one another (because they depend on independent randomnesses), they are unable to leverage the benefit of minimal set cover for incrementality, whereas, our strategy through post-order labeling enables us to leverage this benefit by using only a single group element as an update key. Apart from this crucial difference our scheme also has additional benefits: (i) our ciphertexts are on average 2x smaller than theirs as they used IBE in a black-box manner (both the schemes have $\log(T)/2$ group elements in their ciphertexts on average); (ii) our decryption key contains between 1 to $\log(T)$ group elements, whereas theirs is always $\log(T)$ group elements, making our keys 2x smaller on average.

Additional Relevant Works. Specter et al. [37] add deniability to emails by divulging private signing keys over time from a hierarchical identity-based signature scheme, adapted from the Gentry-Silverberg scheme [24]. Moreover, their hierarchy mimics that of a calendar, and they achieve succintness by allowing a child's key to be derivable from the parent's key. While there is technical similarity, our scheme shows how a binary identity space can enable more efficient tree-based encryption with shorter keys. The scheme by Ning et al. [31] splits a secret into shares, and requires the shareholders to release their shares at a future time or get penalized by a smart contract.

3 TECHNICAL OVERVIEW

We distinguish between update keys, each of which is released at a time epoch, and decryption keys which lets one decrypt all ciphertexts encrypted up to a specific time. Looking ahead, a decrypton key K_τ for time τ is constructed from the update key uk_τ for time τ and the decrypton key $K_{\tau-1}$ for $\tau - 1$.

3.1 Deployment and Operation

For clarity, we first describe the system design with a single server, and then discuss the threshold scenario.

Smart Contract Service. Timed-release encryption is used as a service to smart contracts provided by a trusted time server, who periodically modifies a smart contract with the decryption key for the latest epoch. In particular, the contract is initialized with the decryption key for epoch 0. From then on, once every epoch, the server issues a transaction containing an update key for the latest epoch. The transaction is destined to a special *aggregator contract* that uses the update key to compute the latest decryption key – any application smart contract (e.g. auction) can read the aggregator's most-recent decryption key.

Threshold Setting. To avoid having a single point of failure or trust, we show how to extend to a threshold setting that uses a collection of servers. A setup phase establishes a *lifetime (long-term) secret key* lsk, and generates a corresponding public key lpk. Instead of the whole lsk, the setup phase outputs shares of lsk computed using a t out of n threshold secret sharing scheme [36][5]; here, n denotes the number of servers and t is the corruption threshold, as in t shares are required to reconstruct lsk and any subset of $t-1$ shares reveals no information (in the information-theoretic sense) about lsk. Each server S_i is given a share lsk_i of the whole secret lsk.

Operation. Clients use the public key lpk to encrypt messages, at which point they must also specify a future epoch. During any given epoch τ, each server S_i publishes a *partial* update key $uk_{\tau,i}$. Given any t such partial tokens, the aggregator contract can combine them to attain the *whole* update key uk_τ; the uk_τ is then combined with $K_{\tau-1}$ to compute K_τ. The application contract then uses K_τ to decrypt any ciphertext "locked" to epoch τ or earlier. Note that a single instance of timed-release encryption service can support an arbitrary number of applications. This allows the servers' cost to be amortized; therefore, we must look past simpler schemes that scale poorly, such as having the sender secret-share each message to the servers to be later released to the receiver, for instance.

Observe the following key characteristics:

- The scheme is non-interactive, in that the keys output by the server do not depend on the message or the ciphertext. Moreover, there is no interaction amongst the servers either.
- The whole lsk is never made available to any party.
- Decryption for a ciphertext locked to τ requires a key for epoch $\tau' \geq \tau$, which is available when at least t servers release their shares of the update key for epoch τ' – this ensures that at least one honest server must have waited until epoch τ.

Next we provide technical highlights of our scheme, and defer full details to Section 6. First, we describe the prior IBE based construction, which fails to meet our efficiency requirements. Then, wee describe our main construction, and later show how to thresholdize it. Finally, we mention how to achieve CCA and malicious security.

3.2 Prior IBE-based Constructions

Let us briefly examine how prior works [12, 16] essentially adapt identity-based encryptions (IBE) to the purpose of timed-release encryption. The basic idea is to apply the pairing-based IBE scheme of Boneh and Franklin [4], where each identity is mapped to an epoch. The scheme makes use of source groups[6] $\mathbb{G}$ and $\mathbb{G}_T$ which are both cyclic groups of prime order q, a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, generator $g \in \mathbb{G}$, and hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}$. The scheme essentially works as follows:

- Setup : Sample random $\alpha \leftarrow_\$ Z_q$, and output the lifetime public key $lpk := g^\alpha$, and lifetime secret key $lsk := \alpha$.
- In each epoch, the server outputs update key $uk_\tau := \mathcal{H}(\tau)^\alpha$.
- $\mathsf{Enc}(m, \tau)$ outputs $c := (g^r, m \cdot e(\mathcal{H}(\tau)^r, lpk)$ for $r \leftarrow_\$ Z_q$.

[5]In the threshold setting, the setup phase can also be performed using a distributed key generation protocol [23], in lieu of assuming a trusted dealer, such that no single party learns the lifetime secret key, however we do not place any such demands on the setup phase in our definition and it does not matter that much because it is done only once in the beginning.

[6]For ease of exposition, we assume the symmetric variant of pairings where both the source groups are same – our implementation uses asymmetric pairings for efficiency.

– $\mathrm{Dec}(uk_\tau, c)$ outputs $m := v \cdot e(uk_\tau, u)^{-1}$ where $c = (u, v)$.

Note that these operations correspond closely to the Boneh-Franklin IBE construction [4]. A debilitating property of this scheme is that the keys $\{\mathcal{H}(\tau)^\alpha\}_{\tau \in 1 \dots T}$ are unstructured, in that they cannot be aggregated or compressed (without α). In particular, there is no way to compactly describe a decrypton key at time epoch τ, which would be used to decrypt *any* ciphertext encrypted to a time epoch $\tau' \le \tau$.

3.3 HIBE-based approach

Our initial observation is that keys must mimic the inherent *hierarchy* amongst the epochs: key material for a later epoch must subsume that of earlier epochs, while not revealing any bits of information about the future epochs. This points us towards hierarchical identity-based encryption [24, 25] (HIBE).

In addition to the properties of an IBE scheme, HIBE assumes a partial ordering for the identity-space and derives keys hierarchically. That is, in addition to associating a key with each identity, given identities $id \le id'$, the key for id can be derived from the key for id', via a property called *delegation*. This property is beneficial for our use case, wherein we can define a partial order amongst the epochs, and thus avoid using keys of lower-level epochs if a higher-level update key is already available. In particular, we focus on a particular HIBE construction for a restriced identity space of binary strings, known as Binary Tree Encryption (BTE) [8]. Consider arranging identities or epochs in a binary tree, as illustrated below for $T = 15$ epochs.

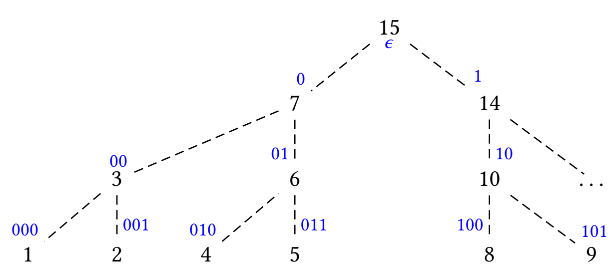


Figure 1: Double labeling of tree with epoch id (via post-order traversal) and path id (binary encoding of path from root node)

Each node is given two labels or identities: the epoch id τ ($0 \le \tau \le T$), and a path id ω containing a binary string that denotes the path to that node from the root (left child being 0 and right child being 1) – we find it convenient to present the construction using the path-based identity, but it is not strictly necessary. The path identities form a *prefix order relation*, where $\omega \le \omega'$ if ω' is a prefix of ω – we denote the root node (or the least upper bound) by the empty string ϵ. The epoch id is assigned via a post-order traversal on the binary tree, which gives us a very useful property explained later.[7]

Consider a non-leaf node with path id ω and epoch id τ. Then, any of its descendant has an epoch id $\tau' < \tau$. Furthermore, ω is a prefix of ω', which is the path id corresponding to τ'. Clearly, a HIBE key (we refer to such keys as update keys) for a node labeled ω can be used to derive a key corresponding to node ω'. This satisfies our requirement because any such epoch τ' is smaller than τ. So, it is sufficient to include a single update key for τ to enable decryption corresponding to any τ' which is the epoch id of a descendant — this enables the desired compression. Note that, the set of descendants, however, do not exhaust all prior epochs, and therefore we need to include more update keys into a decrypton key to enable the time hierarchy we want. However, the tree-structure guarantees that any decrypton key does not contain more than $\log(T)$ update keys.

For instance, we can publish the HIBE key for node 000 in epoch 1, 000 and 001 in epoch 2, 00 in epoch 3, nodes 00 and 010 in epoch 4, and so on. In particular, for a node with path id ω, a decrypton key consists of w update keys, where w denotes the hamming weight of bit-string ω. Therefore, in the worst case a decrypton key consists of $\log(T) + 1$ update keys.[8] Instantiating with a scheme such as BTE, that uses $O(\log(T))$ group elements for one update key, we obtain a scheme where a decrypton key requires $O(\log^2(T))$ group elements in total.

While investigating this, we find that the delegation property – using an id key to derive keys for lower-level identities – of HIBE is not strictly required for our setting (also see Remark 1). In particular, in contrast to the HIBE requirements, our key-generation procedure always has access to the lifetime secret-key lsk (which actually makes it similar to IBE). Our key contribution is to construct a new encryption scheme that supports a hierarchical decryption, in that any decrypton key for time τ can be used to decrypt a ciphertext corresponding to time $\tau' \le \tau$, but does not support key-delegation such as deriving a decrypton key exactly for epoch τ'. Sacrificing the delegation property enables us to have constant size update keys, which are sufficient to "increment" the time bound key from one epoch to the next one. A new trick we use for this purpose is a post-order labeling of the tree (hence calling it a doubly-labeled tree).

Next we provide an overview of our core i-TiRE construction.

3.4 Our i-TiRE scheme

We now present the core aspects of our scheme below (within the box), and give the full details in Sec. 6. We note that our construction is inspired by the BTE construction of Canetti et al. [8]. In what follows, we use path id and epoch id of a node interchangably — as discussed in Sec. 6, this is enabled by an efficient bijective mapping between these two labels. In the following description, we shall use $\omega|_i$ to denote the first i bits of ω, $|\omega|$ to denote the bit-length of ω, and S_ω to denote the id key for node with path id ω.

[7] It is evident later when we present the construction that binary tree structure provides best space efficiency for our scheme. Furthermore, we also explain why post-order traversal is chosen as opposed to pre-order (as chosen in [8]) or in-order ones.

[8] Note that this happens for nodes with path id $111\dots1$.

Lifetime Keys : $lsk = \alpha,\ lpk = g^{\alpha}$ where $\alpha \leftarrow_{\$} Z_q$

Key for path id ω generated with lsk :

$$S_{\omega} = \mathcal{H}(\epsilon)^{\alpha} \cdot \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)^{\alpha}$$

Encryption of M :

$C = (g^{\gamma}, \mathcal{H}(\omega|_1)^{\gamma}, \mathcal{H}(\omega|_2)^{\gamma} \ldots, \mathcal{H}(\omega)^{\gamma}, M \cdot d)$

where $d = e(lpk, \mathcal{H}(\epsilon))^{\gamma} = e(g, \mathcal{H}(\epsilon))^{\alpha\gamma}$

Decryption of C with S_{ω} :

Parse C as $(U_0, U_1, \ldots, U_t, V)$

Output $M = V \cdot d^{-1}$ where $d = \dfrac{e(U_0, S_{\omega})}{\prod_{i=1}^{|\omega|} e(lpk, U_i)}$

The bilinearity property of pairings implies that:

$$\begin{aligned} d &= \frac{e(U_0, S_{\omega})}{\prod_{i=1}^{|\omega|} e(g^{\alpha}, U_i)} \\ &= \frac{e(g^{\gamma}, \mathcal{H}(\epsilon)^{\alpha} \cdot \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)^{\alpha})}{\prod_{i=1}^{|\omega|} e(g^{\alpha}, \mathcal{H}(\omega|_i)^{\gamma})} \\ &= \frac{e(g, \mathcal{H}(\epsilon))^{\alpha\gamma} \cdot \prod_{i=1}^{|\omega|} e(g, \mathcal{H}(\omega|_i))^{\alpha\gamma}}{\prod_{i=1}^{|\omega|} e(g, \mathcal{H}(\omega|_i))^{\alpha\gamma}} \\ &= e(g, \mathcal{H}(\epsilon))^{\alpha\gamma} \end{aligned}$$

The update keys for our example are shown here in red.

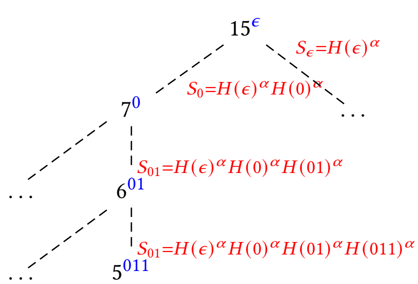


Figure 2: update keys for our doubly-labeled tree

For instance, say that we wish to encrypt for $\omega = 01$, and decrypt first using S_{01} and later using S_0 (note that for 01, the decrypton key consists of two update keys). Encryption of message M for $\omega = 01$ produces the ciphertext:

$$(U_0 = g^{\gamma},\ U_1 = \mathcal{H}(0)^{\gamma},\ U_2 = \mathcal{H}(01)^{\gamma},\\ V = M \cdot e(g^{\alpha}, \mathcal{H}(\epsilon))^{\gamma})$$

Observe that the $e(g^{\alpha}, \mathcal{H}(\epsilon))^{\gamma}$ term acts as a random mask based on the client's chosen randomness γ. Given id key $S_{01} = \mathcal{H}(\epsilon)^{\alpha} \cdot \mathcal{H}(0)^{\alpha} \cdot \mathcal{H}(01)^{\alpha}$, we decrypt the above ciphertext as follows:

$$V \cdot d^{-1} = V \cdot \frac{e(g^{\alpha}, U_1) \cdot e(g^{\alpha}, U_2)}{e(U_0, S_{01})} = M$$

More importantly, due to hierarchical structure, we can also decrypt the same ciphertext using id key $S_0 = \mathcal{H}(\epsilon)^{\alpha} \cdot \mathcal{H}(0)^{\alpha}$:

$$\begin{aligned} V \cdot d^{-1} &= V \cdot \frac{e(g^{\alpha}, U_1)}{e(U_0, S_0)} \\ &= M \cdot e(g^{\alpha}, \mathcal{H}(\epsilon))^{\gamma} \cdot \frac{e(g^{\alpha}, \mathcal{H}(0)^{\gamma})}{e(g^{\gamma}, \mathcal{H}(\epsilon)^{\alpha}) \cdot e(g^{\gamma}, \mathcal{H}(0)^{\alpha})} \\ &= M \end{aligned}$$

We emphasize that $S_{\omega'}$ for any prefix ω' of ω is sufficient to decrypt a ciphertext locked to ω. The intuition behind this scheme is that the ciphertext for ω contains an element corresponding to each prefix of ω, and hence, can be thought of as encrypting to each prefix of ω (or an epoch corresponding to each node from the root to the node for ω). Therefore, when given a key corresponding to a node for any prefix $\omega' > \omega$, we can ignore the remaining elements of the ciphertext (i.e., beyond $U_{|\omega'|} = \mathcal{H}(\omega|_{|\omega'|})^{\gamma}$) and decrypt as if the ciphertext was instead locked to id ω'.

Remark 1 (Comparison with HIBE and IBE). *Delegation means that* anyone *with a key for id ω can derive a key for id ω'. This is useful in the original motivation for HIBE, where a separate party can assume full ability to derive keys in an identity subspace (e.g. a team within a larger organization) with respect to the assigned hierarchical structure. However, in* i-TiRE, all update keys *are issued by the same server, and access to the lifetime secret lsk gives the server ability to compute the key for any epoch/id — this is rather similar to IBE. Therefore, in our case, it suffices to enforce the hierarchy efficiently without providing the ability to delegate.*

Given the above scheme, it is straightforward to construct a i-TiRE scheme: for any epoch τ, the decrypton key consists of $O(\log(T))$ S values, such that their subtrees cover *all epochs* between 1 and τ. For instance, in our running example with $T = 15$, the key for epoch 4 is $K_4 = \{S_{00}, S_{010}\}$, epoch 5 is $K_5 = \{S_{00}, S_{010}, S_{011}\}$, and so on. The final epoch 15 is $K_{15} = \{S_{\epsilon} = \mathcal{H}(\epsilon)^{\alpha}\}$, which simply allows decryption of all ciphertexts encrypted with the lpk. Since each S value is a single group element, our update keys have $O(1)$ size and the decrypton keys have $O(\log(T))$ size (as opposed to $O(\log^2(T))$ in HIBE). We stress that hierarchical decryption is the key enabler here, as it allows us to prune S values of children once a parent node's key can be emitted.

Incremental Updates. When releasing keys sequentially in the order of epochs, the post-order traversal ensures the following fact. The set of S values for any decryption key $K_{\tau+1}$ includes all but one of the S values from K_{τ}; therefore, the server must release only one S value (one group element) as the update key in each epoch when computing keys incrementally.

3.4.1 *Drawbacks of Unbalanced Trees.*

We have a hierarchical structure of a *balanced* binary tree similar to BTE [8]. As explained above, the standard IBE gives a linear structure which fails to achieve our efficiency requirement of compact time-bound keys. Therefore, a tree-like structure seems inevitable in order for an aggregate key to cover a large number of epochs. However, as we have seen, an update key for epoch τ does not cover all keys corresponding to epochs $< \tau$ — this leads to an decrypton key size of $O(\log(T))$. One might wonder whether this blow up can be avoided by instead employing an unbalanced tree: each node in the tree would have a right child which is a leaf, and a left child which branches out further.

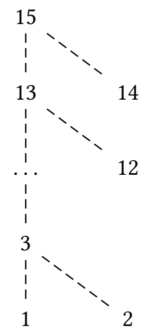


Figure 3: Unbalanced id tree

Such a tree (for $T = 15$) is illustrated to our left. It is easy to see that such a structure indeed supports decrypton keys of $O(1)$ sizes, in that all non-leaf nodes with epoch id $\tau' < \tau$ are contained within the sub-tree rooted at τ. That would mean that an decrypton key could contain as few as two id-keys (two group elements). However, this leads to a blow up in the ciphertext size, rendering it to contain $O(T)$ many group elements — intuitively, this occurs because each ciphertext for a node with path id ω must contain $\Omega(|\omega|)$ group elements when using the above encryption technique with hierarchical decryption. In an unbalanced tree, a path id ω can have up to T bits. Thus, such alternatives fail to achieve our efficiency requirements.

3.4.2 Thresholdizing and CCA security. Since our update keys consist of a single S value, computed by exponentiating the lifetime secret key $lsk = \alpha$ on a known group element, it is simple to compute that when the lifetime secret key is distributed in a manner such that there are n servers holding a (t, n)-threshold secret sharing of lsk. Basically, instead of computing values like $\mathcal{H}(v)^{lsk}$ (for some value v) the i-th server now computes $\mathcal{H}(v)^{lsk_i}$. The client, on receiving any t such values, can combine them using Lagrange reconstruction in the exponent to get the full update key. This step is similar to the threshold computation of PRF as proposed in [2, 29], and it lets us handle up to $t-1$ server compromises. In Section 6.2 we show the extension to the threshold setting. In fact, we consider security against malicious adversaries who can corrupt up to $t-1$ parties. In this setting it is crucial that a client can verify the responses from each server and thus protect against malicious corruption – this is enabled by efficient non-interactive zero-knowledge proofs. In Section 6.3, we outline how to use a variant of the Fujisaki-Okamoto [22] transformation (also used in BTE [8]) to obtain CCA-security. Importantly, these two augmentations can be made independently of each other and hence one can easily combine them to obtain a CCA-secure construction (c.f. Corollary 1) which supports threshold key-generation and is resilient against malicious attacks.

4 DEFINITIONS

4.1 Incremental Timed-Release Encryption (i-TiRE)

Definition 1 (i-TiRE). An *incremental timed-release encryption* (i-TiRE) scheme is a tuple of algorithms (Setup, UKGen,DKGen, Enc, Dec) with the following syntax:

- $\mathsf{Setup}(1^\kappa, T) \to (pp, lpk, lsk)$: On input the security parameter 1^κ and the lifetime duration T (in the number of epochs), Setup generates public parameters pp (to be used by all algorithms that follow), a lifetime public key lpk, a lifetime secret key lsk.
- $\mathsf{UKGen}(lsk, \tau) \to uk_\tau$: On input a lifetime secret key lsk and an epoch $\tau \in \{1, \ldots, T\}$, this algorithm outputs an update key uk_τ specific to the epoch τ.
- $\mathsf{DKGen}(K_{\tau-1}, uk_\tau) \to K_\tau$: On input a decryption key for the (previous) epoch $\tau - 1$ ($\tau \in 1, \ldots, T$) and update key for (current) epoch τ, this algorithms output the decryption key K_τ for the (current) epoch τ.
- $\mathsf{Enc}(lpk, m, \tau) \to c$: encrypts a message m "locked to" epoch τ, using the lifetime public key lpk, and outputs a ciphertext c.
- $\mathsf{Dec}(lpk, K, c) \to m/\bot$: deterministically decrypts the ciphertext c using a decryption key K, returning $\bot$ on failure.

Then, the following condition holds for any $\kappa, T \in \mathbb{N}$. Let $(pp, lpk, lsk) \leftarrow \mathsf{Setup}(1^\kappa, T)$; then, for any message m, any two epochs $\tau, \tau' \in [T]$ for which $\tau \le \tau'$, it satisfies:

(i) *correctness*, that is there exists a negligible function $\mathsf{negl}(\cdot)$ for which the following probability is at least $1 - \mathsf{negl}(\kappa)$:

$$\Pr\Big[m \leftarrow \mathsf{Dec}(lpk, K_{\tau'}, c) \;\Big|\; \begin{array}{c} (pp, lpk, lsk, K_0) \leftarrow \mathsf{Setup}(1^\kappa, T); \\ c \leftarrow \mathsf{Enc}(lpk, m, \tau); \\ uk_1 \leftarrow \mathsf{UKGen}(lsk, 1);\; K_1 \leftarrow \mathsf{DKGen}(K_0, uk_1); \\ \vdots \\ uk_{\tau'} \leftarrow \mathsf{UKGen}(lsk, \tau');\; K_{\tau'} \leftarrow \mathsf{DKGen}(K_{\tau'-1}, uk_{\tau'}) \end{array} \Big]$$

where the probability is over the random coin tosses of the parties involved in Setup, UKGen, DKGen and Enc;

(ii) *efficiency*, that is both $|K_\tau|$ and $|c|$ are proportional to $O(\log(T))$.

(iii) *incrementality*, that is $|uk_\tau|$ is of size $O(1)$.

We define a security game (IND-TR-CCA) for achieving chosen ciphertext security against a "selective" i-TiRE attacker who commits to the epoch to be attacked in advance (before the setup phase). For that reason, we call this attack a *selective-epoch* attack. Our definition is inspired by the security definitions for binary tree encrypion [8].

First, the attacker $\mathcal{A}$ submits her target epoch $\tau^\star$. As is common in CCA games, we allow $\mathcal{A}$ to perform a set of queries both before and after sending the challenge plaintexts. To avoid trivial wins, the game checks whether $\mathcal{A}$ issued a key generation query for any epoch on or after $\tau^\star$, whose result can be used to decrypt for epoch $\tau^\star$. The challenger C responds to a polynomial number of decryption and key generation queries by $\mathcal{A}$, after which $\mathcal{A}$ submits a challenge pair of equal-length messages m_0, m_1; C selects a random bit b and sends $\mathcal{A}$ the encryption of m_b locked to the epoch $\tau^\star$ (selected by $\mathcal{A}$ earlier). After receiving the challenge ciphertext, $\mathcal{A}$ submits another set of decryption and key generation queries, under the constraint that $\mathcal{A}$ is not requesting the decryption of the challenge ciphertext nor is requesting a key for any epoch $\geq \tau^\star$. Finally, $\mathcal{A}$ outputs the guess bit b' and wins if $b' = b$.

Definition 2 (IND-TR-CPA/CCA). A i-TiRE := (Setup, UKGen, UKCombine, Enc, Dec) scheme satisfies *indistinguishability under chosen ciphertext attack* if for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that the *advantage* of $\mathcal{A}$ is given by

$$\left|\Pr\left[\mathsf{CCA}_{\text{i-TiRE},\mathcal{A}}(1^\kappa, 0) = 1\right] - \Pr\left[\mathsf{CCA}_{\text{i-TiRE},\mathcal{A}}(1^\kappa, 1) = 1\right]\right| \leq \mathsf{negl}(\kappa),$$

in a security game CCA which is defined below.

$\mathsf{CCA}_{\text{i-TiRE},\mathcal{A}}(1^\kappa, b)$:

- *Selection.* $\mathcal{A}(1^\kappa, T)$ outputs an epoch $0 \leq \tau^\star \leq T$.
- *Initialization.* Initialize $\tau_{max} := 0$. Run Setup$(1^\kappa, T)$ to get (pp, lpk, lsk, K_0). Give (pp, lpk) to $\mathcal{A}$.
- *Phase 1.* $\mathcal{A}$ adaptively issues a polynomial number of queries, each of one of two types:
  - *Pre-challenge decryption.* In response to $\mathcal{A}$'s decryption query $(\mathtt{Decrypt}, \tau, c)$, C responds by generating the update key K_τ (by running UKGen many times as needed), and using it to decrypt c.
  - *Pre-challenge key derivation.* In response to $\mathcal{A}$'s key derivation query $(\mathtt{Derive}, \tau)$, run UKGen with lsk and τ and return the output to $\mathcal{A}$. Update $\tau_{max} := max(\tau_{max}, \tau)$.
- *Challenge.* $\mathcal{A}$ outputs $(\mathtt{Challenge}, m_0, m_1)$ where $|m_0| = |m_1|$. Give $c^\star \leftarrow \mathsf{Enc}(pp, m_b, \tau^\star)$ to $\mathcal{A}$. Output 1 if $\tau_{max} \geq \tau^\star$.
- *Phase 2.* $\mathcal{A}$ adaptively issues a polynomial number of queries, each of one of two types:
  - *Post-challenge decryption.* Repeat phase 1 but with the following caveat. Only process $\mathcal{A}$'s decryption query $(\mathtt{Decrypt}, \tau, c)$ if $c \neq c^\star$, else return $\perp$ to $\mathcal{A}$.
  - *Post-challenge key derivation.* Repeat phase 1 but with the following caveat: respond to $\mathcal{A}$'s key derivation query $(\mathtt{Derive}, \tau)$ only if $\tau < \tau^\star$else return $\perp$ to $\mathcal{A}$.
- *Guess.* Finally, $\mathcal{A}$ returns a guess b'. Output b'.

When the attacker is prohibited from invoking the decryption oracle, the above definition achieves a weaker guarantee called indistinguishability under chosen plaintext attack or IND-TR-CPA. However, even in IND-TR-CPA, the adversary is given access to the key-derivation oracle. The corresponding experiment is denoted by $\mathsf{CPA}_{\text{i-TiRE},\mathcal{A}}$.

4.2 Threshold i-TiRE

In a threshold i-TiRE scheme there are n parties, each of which holds a share of the lifetime secret key. Therefore, the algorithm to generate the (partial) update keys is now run by a single party using her share of the secret-key instead of the whole secret key. Additionally, there's a (public) combine algorithm which combines the partial update keys to construct the entire update key. Apart from these changes, the syntax remains the same. We consider a t out of n threshold setting where any t $(\leq n)$ partial update keys can be combined to construct the entire update key, but no $t' < t$ partial keys suffice. We provide the syntax below, and omit the formal correctness definition, which can be adjusted straightforwardly. The efficiency and incrementality conditions remain exactly the same. For reader's convenience we highlight the major changes in syntax in *blue*.

A *threshold* incremental timed-release encryption (i-TiRE) scheme is a tuple of algorithms (Setup, PartUKGen, Combine, DKGen, Enc, Dec) where the syntax for algorithms DKGen, Enc, Dec remain unaltered from the previous (non-threshold) definition. So we only provide the syntax for the other algorithms below.

- Setup$(1^\kappa, T, n, t) \to (pp, lpk, (lsk_1, \ldots lsk_n))$: On input the security parameter 1^κ and the lifetime duration T (in the number of epochs), Setup generates public parameters pp (to be used by all algorithms that follow), a life-time public key lpk, n shares of lifetime secret key $(lsk_1, \ldots lsk_n)$. .
- PartUKGen$(lsk_j, \tau) \to uk_{\tau,j}$: On input a share of lifetime secret key lsk_j and an epoch $\tau \in \{1, \ldots, T\}$, this algorithm outputs a partial update key $uk_{\tau,j}$ specific to the epoch τ and party j.
- UKCombine$(uk_{\tau,1}, \ldots, uk_{\tau,t}) \to uk_\tau$ combines t partial update keys into a whole update key.

In the threshold setting the security definition also changes accordingly. In particular, in a t out of n setting, the adversary, in addition to making CPA/CCA queries as elaborated in Definition 2, may also maliciously corrupt up to $t - 1$ parties in the security game. We describe the changed security game below (with the major changes highlighted in *blue* as well).

$\mathsf{IND\text{-}ThTR\text{-}CCA}_{\text{Th-i-TiRE},\mathcal{A}}(1^\kappa, b)$:

- *Selection.* $\mathcal{A}(1^\kappa, T, n, t)$ outputs an epoch $0 \leq \tau^\star \leq T$.
- *Initialization.* Initialize $\tau_{max} := 0$. Run Setup$(1^\kappa, T, n, t)$ to get $(pp, lpk, (lsk_1, \ldots, lsk_n))$. Give (pp, lpk) to $\mathcal{A}$.
- *Corruption.* $\mathcal{A}$ outputs a set of corrupt party's identities $C \subseteq [n]$ such that $|C| < t$. Give lsk_i to $\mathcal{A}$ for all $i \in C$.
- *Phase 1.* $\mathcal{A}$ adaptively issues a polynomial number of queries, each of one of two types:
  - *Pre-challenge decryption.* In response to $\mathcal{A}$'s decryption query $(\mathtt{Decrypt}, \tau, c)$, C responds by generating the update key K_τ (by running UKGen and UKCombine many times in sequence as needed), and using it to decrypt c.
  - *Pre-challenge key derivation.* In response to $\mathcal{A}$'s key derivation query $(\mathtt{Derive}, \tau, j)$ where $j \in [n] \backslash C$, run PartUKGen with lsk_j and τ and return the output to $\mathcal{A}$. Update $\tau_{max} := max(\tau_{max}, \tau)$ only if $(\mathtt{Derive}, \tau, j)$ is asked for at least $t - |C|$ different j values — the challenger can track by storing the queries in a list L_τ for each τ.
- *Challenge.* $\mathcal{A}$ outputs $(\mathtt{Challenge}, m_0, m_1)$ where $|m_0| = |m_1|$. Give $c^\star \leftarrow \mathsf{Enc}(pp, m_b, \tau^\star)$ to $\mathcal{A}$. Output 1 if $\tau_{max} \geq \tau^\star$.
- *Phase 2.* $\mathcal{A}$ adaptively issues a polynomial number of queries, each of one of two types:
  - *Post-challenge decryption.* Repeat phase 1 but with the following caveat. Only process $\mathcal{A}$'s decryption query $(\mathtt{Decrypt}, \tau, c)$ if $c \neq c^\star$, else return $\perp$ to $\mathcal{A}$.
  - *Post-challenge key derivation.* Repeat phase 1 but with the following caveat: respond to $\mathcal{A}$'s key derivation query

(Derive, τ, j) only if either $\tau < \tau^{\star}$ or L_τ has $< t - |C|$ distinct j values, else return $\perp$ to $\mathcal{A}$.

– *Guess.* Finally, $\mathcal{A}$ returns a guess b'. Output b'.

Remark 2 (Adaptive security). *The definition achieves a stronger adaptive security if the "selection" phase takes place after 'corruption" but before the challenge phase. Our construction can be generically transformed to satisfy adaptive security by using complexity leveraging, that is by assuming sub-exponential security of the underlying assumption.*

5 NOTATIONS AND PRIMITIVES

Notation. The set of all binary strings of length ℓ is denoted as $\{0,1\}^\ell$. Sometimes we denote 1^ℓ or 0^ℓ to denote strings of 1 and 0 resp. repeated ℓ times. The output y of a probabilistic algorithm A on input x is denoted by $y \leftarrow A(x)$. For deterministic algorithms sometimes we use $y := A(x)$. Moreover, occasionally we need to explicitly specify the randomness r of a probabilistic algorithm, which is denoted by $y := A(x; r)$. For any bitstring w, we write $w|_i$ to denote the first i bits of w. We denote the empty string by ϵ.

5.1 Bilinear Pairings

Certain elliptic curves have an additional structure, called a *bilinear pairing*. We use the following definitions from [6].

Definition 3. *Let $\mathbb{G}_0$, $\mathbb{G}_1$, $\mathbb{G}_T$ be three cyclic groups of prime order q where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A pairing is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ satisfying the following properties:*

– *bilinear:*

$$\forall u, u' \in \mathbb{G}_0.\ \forall v \in \mathbb{G}_1.\ e(u \cdot u', v) = e(u, v) \cdot e(u', v)$$
$$\forall u \in \mathbb{G}_0.\ \forall v, v' \in \mathbb{G}_1.\ e(u, v \cdot v') = e(u, v) \cdot e(u, v')$$

– *non-degenerate: $g_T =: e(g_0, g_1)$ is a generator of $\mathbb{G}_T$.*

Bilinearity implies the following property:

$$e(g_0^\alpha, g_1^\beta) = e(g_0, g_1)^{\alpha \cdot \beta} = e(g_0^\beta, g_1^\alpha)$$

The decision-BDH assumption states that given random elements g_0^α, $g_0^\beta \in \mathbb{G}_0$ and g_1^α,$g_1^\gamma \in \mathbb{G}_0$, the value $e(g_0, g_1)^{\alpha \cdot \beta \cdot \gamma} \in \mathbb{G}_T$ is indistinguishable from a random element in $\mathbb{G}_T$.

Definition 4. **Attack Game for Decision bilinear Diffie - Hellman (DBDH) assumption***: let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a bilinear pairing where $\mathbb{G}_0$, $\mathbb{G}_1$, $\mathbb{G}_T$ are cyclic groups of prime order q with generators $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$. For a given adversary $\mathcal{A}$, we define two experiments.*

Experiment $b \in \{0,1\}$*:*

The challenger computes

– $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_q$.
– $u_0 \leftarrow g_0^\alpha$, $u_1 \leftarrow g_1^\alpha$, $v_0 \leftarrow g_0^\beta$*, and* $w_1 \leftarrow g_1^\gamma$
– $z^{(0)} \leftarrow e(g_0, g_1)^{\alpha \cdot \beta \cdot \gamma} \in \mathbb{G}_T$, $z^{(1)} \leftarrow e(g_0, g_1)^\delta \in \mathbb{G}_T$

The adversary is given $(u_0, u_1, v_0, w_1, z^{(b)})$ outputs a bit $\hat{b} \in \{0,1\}$. Let W_b be the event that $\mathcal{A}$ outputs 1 in experiment b. We define $\mathcal{A}$'s advantage in solving the DBDH problem as:

$$\mathsf{DBDHadv}[\mathcal{A}, \mathsf{e}] =|\ Pr[W_0] - Pr[W_1]\ |$$

5.2 Secret Sharing

Definition 5 (Shamir's Secret Sharing). *Let p be a prime. An (n, t, p, s)-Shamir's secret sharing scheme is a randomized algorithm* SSS *that on input four integers n, t, p, s, where $0 < t \leq n < p$ and $s \in Z_p$, outputs n shares $s_1, \ldots, s_n \in Z_p$ such that the following two conditions hold for any set $\{i_1, \ldots, i_\ell\}$:*

– *if $\ell \geq t$, there exists fixed (i.e., independent of s) integers $\lambda_1, \ldots, \lambda_\ell \in Z_p$ (a.k.a. Lagrange coefficients) such that $\sum_{j=1}^{\ell} \lambda_j s_{i_j} = s \bmod p$;*
– *if $\ell < t$, the distribution of $(s_{i_1}, \ldots, s_{i_\ell})$ is uniformly random.*

Concretely, Shamir's scheme works as follows. Pick $a_1, \ldots, a_{t-1} \leftarrow_\$ Z_p$. Let $f(x)$ be the polynomial $s + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{t-1} \cdot x^{t-1}$. Then $s_i = f(i)$ for all $i \in [n]$.

5.3 Sigma Protocols

A sigma protocol allows a prover to convince the verifier that a witness satisfies a statement containing arbitrary linear relations (once we take discrete logarithms) in zero-knowledge, i.e., without revealing any other information about the witness. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. We consider statements of the following type:

$$\exists x_1, \ldots, x_n.\ u_1 = \prod_{j=1}^{n} g_{1j}^{x_j} \wedge \ldots \wedge u_m = \prod_{j=1}^{n} g_{mj}^{x_j}$$

Here, a witness is an assignment $(\alpha_1, \ldots, \alpha_n) \in Z_q^n$ to the variables $x_1, \ldots, x_n$ that makes the formula true, while g_{ij} and u_i values are group elements that are known to the verifier (e.g., public values or constants). The protocol between (P, V) for such a relation is as follows:

$$P \to V : u'_1, \ldots, u'_m \in \mathbb{G} \text{ where } u'_i \leftarrow \prod_{j=1}^{n} g_{ij}^{\alpha'_j}, \alpha'_j \leftarrow_\$ Z_q$$
$$V \to P : c \leftarrow_\$ Z_q$$
$$P \to V : \tilde{\alpha}_1, \ldots, \tilde{\alpha}_n \in Z_q \text{ where } \tilde{\alpha}_j = \alpha'_j + \alpha_j c$$
$$V \text{outputs 1 iff} \left(\prod_{j=1}^{n} g_{ij}^{\tilde{\alpha}_j} \stackrel{?}{=} u'_i \cdot u_i^c \right) \text{ for } i = 1, \ldots, m$$

Fiat-Shamir for Sigma protocol. Using the Fiat-Shamir transform, we can convert the Sigma protocol into a non-interactive zero-knowledge proof system as follows. Instead of obtaining the challenge c from the verifier V, the prover P uses $c = H(u_1, g_{1j}, \ldots, u_m, g_{mj}, u'_1, \ldots, u'_m)$, where H is modeled as a random oracle; i.e., we use a hash of the statement and the first message of P as the challenge.

6 OUR CONSTRUCTIONS

Our i-TiRE constructions, though inspired by the so-called Binary-tree encryption [8], are much simpler and more efficient. Our base i-TiRE construction (Fig. 4) satisfies CPA-security. In Sec. 6.2 we show how to augment this construction to a t out of n threshold setting, secure against malicious corruption of up to $t-1$ parties. Finally, we show how to achieve CCA-security in Sec. 6.3 by a variant of Fujisaki-Okamoto transformation [22] analogous to Canetti et al. [8]. Since these two augmentations are orthogonal, it is possible to combine them easily to obtain a threshold i-TiRE construction satisfying CCA security against malicious corruption (c.f. Corollary 1).

Doubly-labeled tree. We use a binary-tree in our construction analogous to BTE [8]. Each node of the tree is labeled with a binary bit-string as follows: let the depth of the tree be d; then the root is labeled with the empty string ϵ, its left child is labeled 0 and the right child is labeled 1; then the entire tree is labeled recursively such that for each node with label $\omega \in \{0,1\}^*$, its left child is labeled by $\omega 0$ and right child by $\omega 1$. Clearly, any node at level $\delta \in \{1,\ldots,d\}$ is labeled with a binary string of length δ, which is equal to the length of the path from the root to this node. These labels of the nodes are called *primary labels*. We refer to a node by its primary label. Additionally, each node is labeled with an integer (referred to as secondary labels), which is assigned through a *post-order traversal* on the tree. Recall that a post-ordered traversal assigns integer labels in an increasing sequence (that is $1, 2, \ldots$) in order left-right-root recursively. So, we can define a bijective mapping $M : \{0,1\}^* \to \mathbb{N}$ which maps the primary labels to the secondary labels. The inverse mapping from the secondary to primary labels is denoted by $M^{-1} : \mathbb{N} \to \{0,1\}^*$. An example is given in Fig. 1. For the lack of a better name we shall refer to this structure by *doubly-labeled tree*.

Left-extended Family. For any node ω in the tree we define its left-extended (similarly right-extended family) family (denoted as $\mathsf{LEF}(\omega)$) as the set which contains node ω plus all nodes that are left children of any node in the path from root to ω, but do not belong to the path themselves. For example, let $\omega = 0100$, then the path from root to ω is the ordered set $(\epsilon, 0, 01, 010, 0100)$. Among them, only the node 0 has a left child, namely 00, which does not belong to the path. So $\mathsf{LEF}(0100)$ consists of nodes $\{00, 0100\}$. Similarly for 111, we have $\mathsf{LEF}(111) = \{0, 10, 110, 111\}$, because every intermediate node of the path $(\epsilon, 1, 11)$ has a left child that does not belong to the path. It is worth noting that the size of $\mathsf{LEF}(\omega)$ is equal to the *hamming weight* of ω plus one. Equivalently $\mathsf{LEF}(\tau)$ can be defined as the same as $\mathsf{LEF}(\omega)$ when $\tau = M(\omega)$.

Remark 3 (An important property). *A very important property is that for any $\tau \in \{1,\ldots,T\}$ if $\omega := M^{-1}(\tau)$ and $\omega' := M^{-1}(\tau+1)$, then the set difference $\mathsf{LEF}(\omega') \setminus \mathsf{LEF}(\omega) = \{\omega'\}$. In other words all but exactly one element, namely ω', of the set $\mathsf{LEF}(\omega')$ is contained in the set $\mathsf{LEF}(\omega)$. This follows from the labeling through post-order traversal. Looking ahead, this fact ensures that the* **incrementality** *property holds in our constructions.*

6.1 CPA-secure i-TiRE

Our construction follows the basic description from Sec. 3, and it is provided in Fig. 4.[9] While it works for both symmetric and asymmetric pairings, the latter provides smaller sized groups $\mathbb{G}_0$ (for the same level of security) (requiring fewer bits for encoding), and also more efficient group and pairing operations. Moreover, we designed our construction so that the elements of the aggregated key (i.e., the S values) are elements of the smaller group $\mathbb{G}_0$, while the public key is an element of $\mathbb{G}_1$. The ciphertext consists of $|\omega|$ elements of $\mathbb{G}_0$ (where $\omega = M^{-1}(\tau)$), one element from $\mathbb{G}_1$, and one element from target group $\mathbb{G}_T$; since a majority of elements of the ciphertext come from $\mathbb{G}_0$, we get a further reduction in our ciphertext size as well.

The following theorem is proved in Appnedix B.

Theorem 1. *Under the decisional BDH assumption (DBDH in Def. 4), there exists an i-TiRE scheme that satisfies* IND-TR-CPA *security as per Def. 2 in the random oracle model.*

6.2 Threshold i-TiRE

Our threshold mechanism is a straightforward adaptation of distributed psuedo-random function [29] for multiplicative prime-order groups. We first recall their mechanism. The PRF functionality being computed collectively can be written as $f_\alpha(x) = \mathcal{H}(x)^\alpha$, where $\mathcal{H} : \{0,1\}^* \to G$ is a hash function (modeled as a random oracle) and the secret key is $\alpha \in \mathbb{Z}_p$. To distribute the evaluation of f, the secret key α must be secret shared between the parties. In the setup phase, a trusted party samples a master key $\alpha \leftarrow_\$ \mathbb{Z}_p$ and uses Shamir's secret sharing scheme [36] (see Def. 5) with a threshold t to create n shares $\alpha_1, \ldots, \alpha_n$ of α. Share α_i is given privately to the server i. We know that for any set of t parties $\{i_1, ..., i_t\} \subseteq [n]$, there exists integers (i.e. Lagrange coefficients) $\lambda_{i_1}, \ldots, \lambda_{i_t} \in \mathbb{Z}_p$ such that $\sum_{j \in \{i_1,\ldots,i_t\}} \alpha_j \lambda_j = \alpha$. Therefore, it holds that

$$\begin{aligned} f_s(x) = \mathcal{H}(x)^\alpha &= \mathcal{H}(x)^{\sum_{j\in\{i_1,\ldots,i_t\}} \lambda_j \alpha_j} \\ &= \prod_{j\in\{i_1,\ldots,i_t\}} \left(\mathcal{H}(x)^{\alpha_j}\right)^{\lambda_j} \end{aligned}$$

which can be computed in a distributed manner, by having each server i produce $H(x)^{\alpha_j}$. Coming back to our construction, we can write S_ω as a combination of values produced by the above DPRF f, as follows:

$$S_\omega = \mathcal{H}(\epsilon)^\alpha \cdot \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)^\alpha = f_\alpha(\epsilon) \cdot \prod_{j=1}^{|\omega|} f_\alpha(\omega|_j)$$

Reconstruction from partial keys leverages the natural homomorphism. Consider any set of t servers $\{i_1, ..., i_t\} \subseteq [n]$, who publish $\{S_{\omega,1}, \ldots, S_{\omega,t}\}$ respectively. Then, we get:

$$\begin{aligned} \prod_{j\in\{i_1,\ldots,i_t\}} S_{\omega,j}^{\lambda_j} &= \prod_{j\in\{i_1,\ldots,i_t\}} \left(\mathcal{H}(\epsilon)^{\alpha_j} \cdot \prod_{k=1}^{|\omega|} \mathcal{H}(\omega|_k)^{\alpha_j}\right)^{\lambda_j} \\ &= \prod_{j\in\{i_1,\ldots,i_t\}} \left(\mathcal{H}(\epsilon)^{\alpha_j\lambda_j} \cdot \prod_{k=1}^{|\omega|} \mathcal{H}(\omega|_k)^{\alpha_j\lambda_j}\right) \\ &= \mathcal{H}(\epsilon)^\alpha \cdot \prod_{k=1}^{|\omega|} \left(\prod_{j\in\{i_1,\ldots,i_t\}} \mathcal{H}(\omega|_k)^{\alpha_j\lambda_j}\right) \\ &= \mathcal{H}(\epsilon)^\alpha \cdot \prod_{k=1}^{|\omega|} \mathcal{H}(\omega|_k)^\alpha = S_\omega \end{aligned}$$

Furthermore, in this setting to protect against malicious attacker each party needs to publish a NIZK proof (specifically, Schnorr's proof [14, 35] via the Fiat-Shamir transform [21]) to prove the key's validity. For provable security, we use trapdoor commitments to commit to secret key shares of parties and generate NIZKs with

[9] For notational convenience we assume that the UKCombine algorithm works with responses from the first t servers, as opposed to any t servers. The generalization is straightforward.

Ingredients

* Let $\mathbb{G}_0$, $\mathbb{G}_1$, and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order q such that there exists a bilinear pairing $e\colon \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ that is efficiently computable and non-degenerate. Let $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ be generators of the respective groups.
* Hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{G}_0$ modeled as a random oracle
* A doubly-labeled tree Γ of depth d such that $T = 2^d - 1$

CPA-secure i-TiRE

- $\underline{\mathsf{Setup}(1^\kappa, T) \to (pp, lpk, lsk)}$:
  Choose uniform random $\alpha \leftarrow_\$ Z_q$. Then, set $pp := (\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, q, \mathcal{H}, \Gamma)$; $lsk := \alpha$; ; $lpk := g_1^\alpha$.
- $\underline{\mathsf{UKGen}(lsk, \tau) \to uk_\tau}$:
  Parse $\alpha := lsk$. Let $\omega := M^{-1}(\tau)$. Then, compute:
  - $S_\tau := \left(\mathcal{H}(\epsilon) \prod_{k=1}^{\ell} \mathcal{H}(\omega|_k)\right)^\alpha$, where $\ell = |\omega|$

  Output $uk_\tau := (\tau, S_\tau)$.
- $\underline{\mathsf{DKGen}(K_{\tau-1}, uk_\tau) \to K_\tau}$:
  If $\tau = 1$, then $K_1 := ek_1$. Otherwise, let $(\omega_1, \omega_2, \ldots, \omega_\ell) := \mathsf{LEF}(\tau - 1) \cap \mathsf{LEF}(\tau)$. For all $i \in [\ell]$, let $\tau_i := M(\omega_i)$.
  Output $K_\tau := \{uk_{\tau_1}, \ldots uk_{\tau_\ell}, uk_\tau\}$.
- $\underline{\mathsf{Enc}(lpk, m, \tau) \to c}$:
  Let $\omega = M^{-1}(\tau)$. Sample uniform random $r \leftarrow_\$ Z_q$ and then compute:
  - $c_1 := (\tau, g_1^r, \mathcal{H}(\omega|_1)^r, \mathcal{H}(\omega|_2)^r, \ldots, \mathcal{H}(\omega)^r)$;
  - $c_2 := m \cdot e(\mathcal{H}(\epsilon)^r, lpk)$;

  Output $c = (c_1, c_2)$
- $\underline{\mathsf{Dec}(lpk, K, c) =: m/\perp}$:
  Parse c as (c_1, c_2) and then:
  - parse $c_1 := (\tau', R, h_1, \ldots, h_\ell)$;
  - parse $((\tau_1, S_1), \ldots, (\tau_{\eta+1}, S_{\eta+1})) := K$.
  - if $\tau' > \tau_{\eta+1}$ then output $\perp$, else go to the next step;
  - identify the unique (τ_i, S_i) such that either $\tau_i = \tau'$ or $\omega_i := M^{-1}(\tau_i)$ is a prefix of $\omega' := M^{-1}(\tau')$;
  - set $d := e(S_i, R) \cdot (\prod_{i=1}^{\ell_i} e(h_i, lpk))^{-1}$ where $\ell_i := |\omega_i|$;

  Output $m := c_2 \cdot d^{-1}$

Figure 4: Our CPA-secure i-TiRE construction

respect to these commitments, in lieu of simply proving correctness with respect to the public key lpk – since the adversary is allowed to corrupt parties after obtaining the public parameters output by Setup, we make use of trapdoor commitments to let the simulator open the commitments to different values using a trapdoor. Correctness follows from the extractability property of the NIZK scheme and the binding property of the commitment scheme.

Algorithms for our threshold i-TiRE scheme are described in Figure 5, where the major changes from the previous construction are highlighted in blue. The algorithms DKGen, Enc and Dec algorithms remain the same, so we omit mentioning them. We need some additional ingredients:

- A trapdoor commitment scheme ($\mathsf{Setup}_{\mathsf{com}}$, Commit) (Def. 7).
- Another hash function $\mathcal{H}' : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$ modeled as a random oracle (within the NIZK).
- A SS-NIZK := ($\mathsf{Prove}^{\mathcal{H}'}$, $\mathsf{Verify}^{\mathcal{H}'}$) (Def. 8).

The following theorem is proved in Appendix B.

Theorem 2. *Under the decisional BDH assumption (DBDH in Def. 4), there exists a threshold* i*-TiRE scheme that satisfies* IND-TR-CPA *security against malicious adversary in the random oracle model.*

Fig. 5 also defines our concrete instantiation of the trapdoor commitment and NIZK proofs. We use Pedersen commitments (using independent generators $g, h \in \mathbb{G}_0$, whose discrete log is the trapdoor), and Schnorr-style proofs (more generally, sigma protocols (see Sec. 5.3)) made non-interactive using the Fiat-Shamir transformation in the random oracle model. The Setup phase outputs a commitment γ_i to each share α_i using randomness ρ_i.

Threshold i-TiRE

- $\underline{\mathsf{Setup}(1^\kappa, T, n, t) \to (pp, lpk, (lsk_1, \ldots, lsk_n))}$: Choose uniform random $\alpha \leftarrow_\$ Z_q$. Let $lsk_{:=}\alpha$, $lpk := g^\alpha$ and run $(\alpha_1, \ldots, \alpha_n) := \mathsf{SSS}_{n,t,q}(\alpha)$. Run $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$ to get pp_{com}. Sample uniform random ρ_i and compute $\gamma_i := \mathsf{Commit}(pp_{\mathsf{com}}, \alpha_i; \rho_i)$. Then set: $pp := (\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, q, \mathcal{H}, \Gamma, , pp_{\mathsf{com}}, \gamma_1, \ldots, \gamma_n)$ and $lsk_i := (\alpha_i, \rho_i)$.
- $\underline{\mathsf{PartUKGen}(lsk_j, \tau) \to uk_{\tau,j}}$: Parse $\alpha_j := lsk_j$. Let $\omega := M^{-1}(\tau)$ and $\ell = |\omega|$, then
  - Compute $w_0 := \mathcal{H}(\epsilon)$, for $k \in \{1, \ldots, \ell\}$ $w_k := \mathcal{H}(\omega|_k)$.
  - Compute $S_{\tau,j} := (\prod_{k=0}^{\ell} w_k)^{\alpha_j}$.
  - Run $\mathsf{Prove}^{\mathcal{H}'}$ for the language $\{\exists\ \alpha, \rho$ s.t. $S_{\tau,j} = (\prod_{k=0}^{\ell} w_k)^\alpha \wedge \gamma = \mathsf{Commit}(pp_{\mathsf{com}}, \alpha; \rho)\}$ with statement $(S_{\tau,j}, w_0, w_1, \ldots, w_\ell, \gamma_j)$ and witness (α_j, ρ_j) to obtain a proof π_j

  Output $ek_{\tau,j} := (\tau, S_{\tau,j}, \pi_j)$.
- $\underline{\mathsf{UKCombine}(uk_{\tau,1}, \ldots, uk_{\tau,t}) =: uk_\tau/\perp}$: Parse each $uk_{\tau,j} := (\tau, S_{\tau,j}, \pi_j)$, then compute $w_0 := \mathcal{H}(\epsilon)$, for $k \in \{1, \ldots, \ell\}$ and $w_k := \mathcal{H}(\omega|_k)$ where $\omega = M^{-1}(\tau)$. Then, first check whether proof π_j verifies with respect to the statements $(S_{\tau,j}, \tau, \gamma_j)$, if not then output $\perp$, otherwise use Lagrange coefficients $\lambda_j \in Z_q$ to compute $S_\tau := \prod_{j \in [t]} S_{\tau,j}^{\lambda_j}$. Output $uk_\tau := (\tau, S_\tau)$.

Concrete instantiation

- $\mathsf{Setup}_{\mathsf{com}}(1^\kappa)$: Sample generator $h \leftarrow_\$ \mathbb{G}_0$, and output $pp_{\mathsf{com}} =: (g_0, h)$.
- $\mathsf{Commit}(pp_{\mathsf{com}} = (g, h), \alpha; \rho)$: output $g^\alpha \cdot h^\rho$
- $\mathsf{Prove}^{\mathcal{H}'}(\{\exists\ \alpha, \rho.\ S = (\prod_{k=0}^{\ell_i} w_k)^\alpha \wedge \gamma = g^\alpha \cdot h^\rho\}$ with statement $(S, w_0, w_1, \ldots, w_{\ell_i}, \gamma)$ and witness (α, ρ):
  Let $w = \prod_{k=0}^{\ell_i} w_k$. Sample $v, v' \leftarrow_\$ Z_p$ and set $t := w^v$, $t' := g^v \cdot h^{v'}$. Compute $c := \mathcal{H}'(g, h, S, w_0, w_1, \ldots, w_{\ell_i}, \gamma, t, t')$. Let $u := v - c \cdot s$ and $u' := v' - c \cdot r$. Output proof $\pi = (c, u, u')$.
- $\mathsf{Verify}(\pi = (c, u, u'))$ for statement $(S, w_0, w_1, \ldots, w_{\ell_i}, \gamma)$:
  Compute $t := w^u \cdot S^c$, $t_i' := g^u \cdot h^{u'} \cdot \gamma^c$ and output 1 iff $c = \mathcal{H}'(g, h, S, w_0, w_1, \ldots, w_{\ell_i}, \gamma, t, t')$.

Figure 5: Changes for the Threshold i-TiRE construction

Concretely, server i proves the following statement for ω:

$$\exists \alpha_i, \rho_i . \gamma_i := g^{\alpha_i} \cdot h^{\rho_i} \ \wedge\ S_{\omega,i} = \left(\mathcal{H}(\epsilon) \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)\right)^{\alpha_i}$$

We emphasize that our proof contains 3 field elements of Z_q (where q is the order of group $\mathbb{G}_0$), and its size is independent of the bit-length of ω. The reason is that even though $S_{\omega,i}$ is a product of $|\omega|$ terms, it can be written as x^{α_i}, where $x = \mathcal{H}(\epsilon) \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)$ is one group element. Therefore, decryption keys are of size $\Theta(log(T))$ even in the verifiable construction.

6.3 CCA-security

Our CPA-secure construction achieves IND-TR-CCA-security by using a variant of Fujisaki-Okamoto [22] transformation, similar to the BTE construction of Canetti et al. [8]. Remarkably the only changes we need to make are in the Enc and Dec algorithms. Therefore, one can easily deploy these changes together with the augmentation needed for threshold setting as discussed in the previous subsection, thereby achieving a threshold construction satisfying security against malicious corruption and IND-TR-CCA security. To that end, we will need more ingredients:

- A symmetric-key encryption scheme (SE.Enc, SE.Dec) that takes $\{0,1\}^\kappa$ bit key.
- Two hash functions $\mathcal{H}_1 : \mathbb{G}_T \to \{0,1\}^\kappa$ and $\mathcal{H}_2 : \mathbb{G}_T \times \{0,1\}^* \times \{0,1\}^* \to Z_q$ modeled as random oracles.

We describe the changed Enc′ and Dec′ algorithms below. Those are generic extensions of the algorithms Enc and Dec respectively from the base construction.

1. $\mathsf{Enc}'(lpk, m, \tau) \to c$: Let $\omega := M^{-1}(\tau)$. Then:
   - Sample uniform random $s \leftarrow_\$ \mathbb{G}_T$.
   - Compute $c_1 \leftarrow \mathsf{Enc}(lpk, s, \tau; \mathcal{H}_2(s, \omega, m))$.
   - Compute $c_2 \leftarrow \mathsf{SE.Enc}(\mathcal{H}_1(s), m)$ where $\mathcal{H}_1(s)$ is used as the key.
   - Set $c := (\tau, c_1, c_2)$.
2. $\mathsf{Dec}'(lpk, K, c) \to m/\perp$: Parse $(\tau, c_2, c_2) := c$ and let $\omega := M^{-1}(\tau)$ then:
   - Compute $s := \mathsf{Dec}(c_1)$.
   - Use $\mathcal{H}_1(s)$ as the key to decrypt $m := \mathsf{SE.Dec}(\mathcal{H}_1(s), c_2)$.
   - Then, re-encrypt with $\mathcal{H}_2(s, \omega, m)$ to check whether $c_1 = \mathsf{Enc}(lpk, s, \tau; \mathcal{H}_2(s, \omega, m))$. If the check succeeds then output m otherwise output $\perp$.

The following theorem is proved in Appendix B.

Theorem 3. *Under the decisional BDH assumption (DBDH in Def. 4), there exists an* i*-TiRE scheme that satisfies* IND-TR-CCA *security as per Def. 2 in the random oracle model.*

Combining Theorem 2 with the above theorem we get the following corollary immediately.

Corollary 1. *Under the decisional BDH assumption (DBDH in Def. 4), there exists a threshold* i*-TiRE scheme that satisfies* IND-TR-CCA *security against malicious adversary in the random oracle model.*

7 IMPLEMENTATION AND EVALUATION

We measure several attributes of our threshold i-TiRE scheme, including the size of decrypton keys, size of ciphertexts, and the running time of the individual algorithms. We implemented it in Go, using the BLS12-381 curve [7], and released it open source at https://github.com/gotatle/tatle. Benchmarks were run on a Macbook Pro with a 2.6 GHz Intel Core i7 CPU and 16 GB RAM.

7.1 Update Key Size

Due to incremental updates, a server only publishes an update key comprising 1 group element from $\mathbb{G}_0$, which is of length 48 bytes, when serialized in binary form. Contrast this with [32], where update keys are of $log(T)$ size: 0.48 KB, 0.72 KB, 0.96 KB, and 1.44 KB for $T = 2^{10}, 2^{15}, 2^{20}, 2^{30}$, respectively.

7.2 Decrypton Key Size

We measure the size of the decrypton key (produced by DKGen) as a function of the lifetime T. This metric is indicative of the on-chain storage required by the smart contract to maintain the decrypton key.

Recall that the size of the key K_τ for epoch $\tau < T$ depends on the number of tree nodes required to cover the range of epochs from 1 to τ. For that reason, we get a range of key sizes within a lifetime T. We expect $log(T)$ number of nodes in the worst case, and each node has an associated S value in K_τ – each S value is an element of $\mathbb{G}_0$ of length 48 bytes. So, we collect key sizes for the entire range of epochs in a lifetime, and report the max and the average statistics (which unsurprisingly ends up being half of the max size). We also report the key size for both maliciously-secure and semi-honest settings, with the distinction being that the maliciously-secure scheme has a NIZK proof (containing 3 field elements of 32 bytes each) alongside each S value.

The results are in Table 1. Our keys grow logarithmically in T, and it is under 2 KB in the semi-honest and 5 KB in the malicious setting for a lifetime of 2^{30} epochs. On average, our keys are half the size of those in [32] (denoted TSE in Table 1, though no implementation is reported in their work), as their keys are always of size $log(T)$. Had we used an IBE-based scheme, such as [12, 16], decrypton keys would have grown linearly with the number of epochs T, and they end up being prohibitively large (in the order of gigabytes) for even modest sized lifetimes.

epochs	stat	Semi-Honest		Malicious	
		i-TiRE	TSE	i-TiRE	TSE
2^{10}	max	0.480	0.480	1.6	-
	avg	0.240	0.480	0.8	-
2^{15}	max	0.720	0.720	2.4	-
	avg	0.360	0.720	1.2	-
2^{20}	max	0.960	0.960	3.2	-
	avg	0.480	0.960	1.6	-
2^{30}	max	1.440	1.440	4.8	-
	avg	0.720	1.440	2.4	-

Table 1: Key size (in KBs)

7.3 Ciphertext Size

We report the ciphertext size, which can be treated as the overhead from ciphertext expansion when encrypting a binary string in our CCA construction (see Sec. 6.3).

Similar to the case with key size, different epochs produce ciphertexts of varying size, depending on the position of the node in the tree – recall that for path-based identity ω of the node labelled τ, a ciphertext locked to epoch τ will have $|\omega|$ group elements from $\mathbb{G}_0$ (48 bytes each), one element from $\mathbb{G}_1$ (96 bytes), and one element from $\mathbb{G}_T$ (384 bytes). Table 2 reports the min, max, and average statistics over the ciphertext sizes across the lifetime, for various values of T. One can observe that ciphertexts grow logarithmically in the lifetime T.

epochs	2^{10}	2^{15}	2^{20}	2^{30}
min	0.576	0.576	0.576	0.576
max	1.088	1.408	1.728	2.368
avg	1.025	1.344	1.664	2.304

Table 2: Ciphertext Expansion (in KBs)

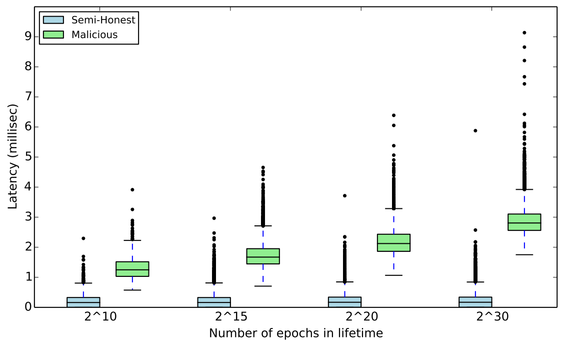


Figure 6: Running Time of UKGen

7.4 Running Time

We report the running times of the UKGen algorithm run by the server, and the Enc and Dec algorithms run by the client (using a whole aggregate key).

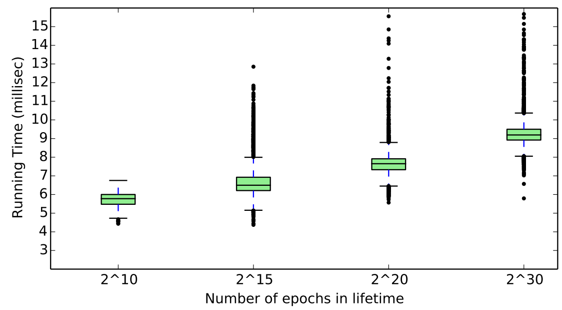
7.4.1 *Key Generation.*

Caching Optimization. We implement an obvious caching optimization when running UKGen for consecutive epochs. For any node with id ω, S_ω is defined recursively as $S_{\omega'} \cdot \mathcal{H}(\omega)^\alpha$ (where ω' denotes the parent of ω). So, if we have cached the S value of any parent node, we can avoid recomputing several group operations. To that end, we maintain a cache comprising S values for each node along the path from the root node to ω, and remove S values of nodes that are no longer needed (because we will never output it in a key nor compute its child in future). Consider Fig. 1; in epoch 4 for instance, we remove S_{000} and S_{001} from the cache, and add S_{01} and S_{010}. We still compute $\Theta(log(T))$ new S values in the worst case (e.g. in epoch 8, where fresh S values must be computed along the entire path from the root). However, observe that intermediate nodes never compute fresh S values, since a leaf node would have already computed the necessary S values. In general, we find a significant drop in the required computation across a large fraction of the nodes. Note that the cache never exceeds the tree's height, so it is at most 64 * $\lceil log(T) \rceil$ bytes (2 KB for 2^{30} epochs).

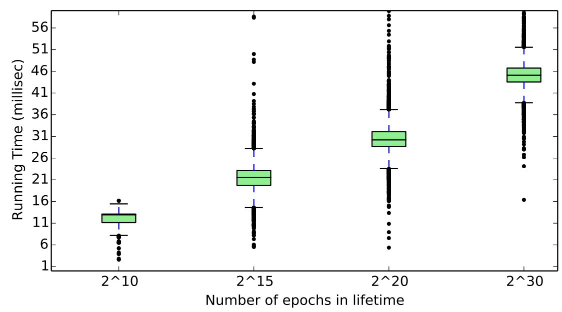
Running Time of Key Generation. Fig. 6 shows the distribution of running times. Most UKGen evaluations terminate under 1 ms due to caching, but we can observe the $\Theta(log(T))$ worst case running time in the outliers. The quantiles grow with T in the malicious execution as the NIZK proofs do not benefit from caching in the same manner as the S values.

7.4.2 *Running Time of Encryption and Decryption.*

Like other metrics, the running time for each Enc and Dec operation depends on the depth of the node – ciphertext includes a group element for each node along the path from the root node – so we plot the distribution attained from 10^6 trials, with each trial operating over a random epoch. The results are shown in Fig. 7.



(a) Running Time of Enc



(b) Running Time of Dec

Figure 7: Distribution of Running Time of Enc **and** Dec **operations**

8 CONCLUSION

We put forward a new timed-release encryption scheme with a crucial *incrementality* property – this enables applications such as performing sealed bid auction over blockchains. Both the decryption key and the ciphertext size of our scheme are proportional to $\log(T)$, where T is the lifetime of the system. Moreover, we show how to strengthen our scheme to a threshold setting, which is secure against malicious adversary and also provides CCA-security.

REFERENCES

[1] Now Released (Fall 2010): Autobiography of Mark Twain, Volume 1. https://www.marktwainproject.org/about_absample.shtml.

[2] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1993–2010. ACM Press, October 2018.

[3] Shashank Agrawal, Payman Mohassel, Pratyay Mukherjee, and Peter Rindal. DiSE: Distributed symmetric-key encryption. Cryptology ePrint Archive, Report 2018/727, 2018. https://eprint.iacr.org/2018/727.

[4] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, August 2001.

[5] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[6] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. Manuscript, 2020. https://toc.cryptobook.us/.

[7] Sean Bowe. Bls12-381: New zk-snark elliptic curve construction. https://electriccoin.co/blog/new-snark-curve/, 2017.

[8] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Heidelberg, May 2003.

[9] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. Cryptology ePrint Archive, Report 2003/083, 2003. http://eprint.iacr.org/2003/083.

[10] Julien Cathalo, Benoît Libert, and Jean-Jacques Quisquater. Efficient and non-interactive timed-release encryption. In Sihan Qing, Wenbo Mao, Javier López, and Guilin Wang, editors, *ICICS 05*, volume 3783 of *LNCS*, pages 291–303. Springer, Heidelberg, December 2005.

[11] Konstantinos Chalkias, Dimitrios Hristu-Varsakelis, and George Stephanides. Improved anonymous timed-release encryption. In Joachim Biskup and Javier López, editors, *ESORICS 2007*, volume 4734 of *LNCS*, pages 311–326. Springer, Heidelberg, September 2007.

[12] Aldar C. F. Chan and Ian F. Blake. Scalable, server-passive, user-anonymous timed release cryptography. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems*, ICDCS '05, page 504–513, USA, 2005. IEEE Computer Society.

[13] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings – the role of Ψ revisited. Cryptology ePrint Archive, Report 2009/480, 2009. http://eprint.iacr.org/2009/480.

[14] David Chaum and Hans Van Antwerpen. Undeniable signatures. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 212–216. Springer, Heidelberg, August 1990.

[15] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 191–205. Springer, Heidelberg, February / March 2006.

[16] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11(2), May 2008.

[17] Gwangbae Choi and Serge Vaudenay. Timed-release encryption with master time bound key (extended). *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 10(4):88–108, 2019.

[18] Mihai Christodorescu, Sivanarayana Gaddam, Pratyay Mukherjee, and Peter Rindal. Amortized threshold symmetric-key encryption. To appear in ACM CCS 2021, 2021. via personal communication.

[19] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 74–89. Springer, Heidelberg, May 1999.

[20] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012.

[21] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[22] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

[23] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[24] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 548–566. Springer, Heidelberg, December 2002.

[25] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 466–481. Springer, Heidelberg, April / May 2002.

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

[27] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 39–50. Springer, Heidelberg, August 2011.

[28] T.C. May. Timed-release crypto. http://www.hks.net.cpunks/cpunks-0/1460.html, 2021.

[29] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 327–346. Springer, Heidelberg, May 1999.

[30] Shutter Network. Shutter – in-depth explanation of how we prevent front running. https://blog.shutter.network/shutter-in-depth-explanation-of-how-we-prevent-frontrunning/, 2021.

[31] Jianting Ning, Hung Dang, Ruomu Hou, and Ee-Chien Chang. Keeping time-release secrets through smart contracts. Cryptology ePrint Archive, Report 2018/1166, 2018. https://eprint.iacr.org/2018/1166.

[32] Kenneth G. Paterson and Elizabeth A. Quaglia. Time-specific encryption. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 1–16. Springer, Heidelberg, September 2010.

[33] Michael O Rabin and Christopher Thorpe. Time-lapse cryptography. 2006.

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

[35] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

[36] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[37] Michael Specter, Sunoo Park, and Matthew Green. Keyforge: Mitigating email breaches with forward-forgeable signatures, 2021.

[38] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

Appendix

A ADDITIONAL PRIMITIVES

A.1 Commitment

Definition 6. *A (non-interactive) commitment scheme* Σ *consists of two PPT algorithms* $(\mathsf{Setup_{com}}, \mathsf{Commit})$ *which satisfy hiding and binding properties:*

- $\mathsf{Setup_{com}}(1^\kappa) \to pp_{\mathsf{com}}$: *It takes the security parameter as input, and outputs some public parameters.*
- $\mathsf{Commit}(m, pp_{\mathsf{com}}; r) =: \alpha$: *It takes a message m, public parameters* pp_{com} *and randomness r as inputs, and outputs a commitment* α.

Hiding. A commitment scheme $\Sigma = (\mathsf{Setup_{com}}, \mathsf{Commit})$ is hiding if for all PPT adversaries $\mathcal{A}$, all messages m_0, m_1, there exists a negligible function negl such that for $pp_{\mathsf{com}} \leftarrow \mathsf{Setup_{com}}(1^\kappa)$,

$$|\Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Commit}(m_0, pp_{\mathsf{com}}; r_0)) = 1] - \Pr[\mathcal{A}(pp_{\mathsf{com}}, \mathsf{Commit}(m_1, pp_{\mathsf{com}}; r_1)) = 1]| \leq \mathsf{negl}(\kappa),$$

where the probability is over the randomness of $\mathsf{Setup_{com}}$, random choice of r_0 and r_1, and the coin tosses of $\mathcal{A}$.

Binding. A commitment scheme $\Sigma = (\mathsf{Setup_{com}}, \mathsf{Commit})$ is binding if for all PPT adversaries $\mathcal{A}$, if $\mathcal{A}$ outputs m_0, m_1, r_0 and r_1 ($(m_0, r_0) \neq (m_1, r_1)$) given $pp_{\mathsf{com}} \leftarrow \mathsf{Setup_{com}}(1^\kappa)$, then there exists a negligible function negl such that

$$\Pr[\mathsf{Commit}(m_0, pp_{\mathsf{com}}; r_0) = \mathsf{Commit}(m_1, pp_{\mathsf{com}}; r_1)] \leq \mathsf{negl}(\kappa)$$

where the probability is over the randomness of $\mathsf{Setup_{com}}$ and the coin tosses of $\mathcal{A}$.

Definition 7 (Trapdoor (Non-interactive) Commitments.). *Let* $\Sigma = (\mathsf{Setup_{com}}, \mathsf{Commit})$ *be a (non-interactive) commitment scheme. A trapdoor commitment scheme has two more PPT algorithms* SimSetup *and* SimOpen*:*

- $\mathsf{SimSetup}(1^\kappa) \to (pp_{\mathsf{com}}, \tau_{\mathsf{com}})$: *It takes the security parameter as input, and outputs public parameters* pp_{com} *and a trapdoor* τ_{com}.
- $\mathsf{SimOpen}(pp_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r)) =: r'$: *It takes the public parameters* pp_{com}, *the trapdoor* τ_{com}, *a message* m' *and a message-randomness pair* (m, r), *and outputs a randomness* r'.

For every (m, r) *and* m', *there exists a negligible function* negl *such that* $pp_{\mathsf{com}} \approx_{stat} pp'_{\mathsf{com}}$, *where* $pp_{\mathsf{com}} \leftarrow \mathsf{Setup_{com}}(1^\kappa)$ *and* $(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa)$*; and*

$$\Pr\left[\mathsf{Commit}(m, pp'_{\mathsf{com}}; r) = \mathsf{Commit}(m', pp'_{\mathsf{com}}; r')\right] \geq 1 - \mathsf{negl}(\kappa)$$

where $r' := \mathsf{SimOpen}(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}, m', (m, r))$ *and* $(pp'_{\mathsf{com}}, \tau_{\mathsf{com}}) \leftarrow \mathsf{SimSetup}(1^\kappa)$.

Clearly, a trapdoor commitment can be binding against PPT adversaries only.

A.1.1 Concrete instantiations. Practical commitment schemes can be instantiated under various settings:

Random oracle. In the random oracle model, a commitment to a message m is simply the hash of m together with a randomly chosen string of length r of an appropriate length.

DLOG assumption. A popular commitment scheme secure under DLOG is Pedersen commitment. Here, $\mathsf{Setup_{com}}(1^\kappa)$ outputs the description of a (multiplicative) group G of prime order $p = \Theta(\kappa)$ (in which DLOG holds) and two randomly and independently chosen generators g, h. If $\mathcal{H} : \{0,1\}^* \to Z_p$ is a collision-resistant hash function, then a commitment to a message m is given by $g^{\mathcal{H}(m)} \cdot h^r$, where $r \leftarrow_\$ Z_p$. A trapdoor is simply the discrete log of h with respect to g. In other words, SimSetup picks a random generator g, a random integer a in $Z_p^\star$ and sets h to be g^a. Given (m, r), m' and a, SimOpen outputs $[(\mathcal{H}(m) - \mathcal{H}(m'))/a] + r$. It is easy to check that commitment to m with randomness r is equal to the commitment to m' with randomness r'.

A.2 Non-interactive Zero-knowledge

Let R be an efficiently computable binary relation. For pairs $(s, w) \in R$, we refer to s as the statement and w as the witness. Let L be the language of statements in R, i.e. $L = \{s : \exists w \text{ such that } R(s, w) = 1\}$. We define non-interactive zero-knowledge (NIZK) arguments of knowledge in the random oracle model based on the work of Faust et al. [20].

Definition 8 (NIZK Argument of Knowledge). *Let* $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{\mathsf{poly}(\kappa)}$ *be a hash function modeled as a random oracle. A* NIZK *for a binary relation R consists of two PPT algorithms* Prove *and* Verify *with oracle access to* $\mathcal{H}$ *defined as follows:*

- $\mathsf{Prove}^{\mathcal{H}}(s, w)$ *takes as input a statement s and a witness w, and outputs a proof* π *if* $(s, w) \in R$ *and* $\perp$ *otherwise.*
- $\mathsf{Verify}^{\mathcal{H}}(s, \pi)$ *takes as input a statement s and a candidate proof* π, *and outputs a bit* $b \in \{0,1\}$ *denoting acceptance or rejection.*

These two algorithms must satisfy the following properties:

- ***Perfect completeness****: For any* $(s, w) \in R$,
$$\Pr\left[\mathsf{Verify}^{\mathcal{H}}(s, \pi) = 1 \mid \pi \leftarrow \mathsf{Prove}^{\mathcal{H}}(s, w)\right] = 1.$$
- ***Zero-knowledge****: There must exist a pair of PPT simulators* $(\mathcal{S}_1, \mathcal{S}_2)$ *such that for all PPT adversary* $\mathcal{A}$,
$$\left|\Pr[\mathcal{A}^{\mathcal{H}, \mathsf{Prove}^{\mathcal{H}}}(1^\kappa) = 1] - \Pr[\mathcal{A}^{\mathcal{S}_1(\cdot), \mathcal{S}'_2(\cdot,\cdot)}(1^\kappa) = 1]\right| \leq \mathsf{negl}(\kappa)$$
for some negligible function negl, *where*
  - $\mathcal{S}_1$ *simulates the random oracle* $\mathcal{H}$;
  - $\mathcal{S}'_2$ *returns a simulated proof* $\pi \leftarrow \mathcal{S}_2(s)$ *on input* (s, w) *if* $(s, w) \in R$ *and* $\perp$ *otherwise;*
  - $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *share states.*
- ***Argument of knowledge****: There must exist a PPT simulator* $\mathcal{S}_1$ *such that for all PPT adversary* $\mathcal{A}$, *there exists a PPT extractor* $\mathcal{E}^{\mathcal{A}}$ *such that*
$$\Pr\left[(s, w) \notin R \text{ and } \mathsf{Verify}^{\mathcal{H}}(s, \pi) = 1 \mid (s, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1(\cdot)}(1^\kappa); w \leftarrow \mathcal{E}^{\mathcal{A}}(s, \pi, Q)\right] \leq \mathsf{negl}(\kappa)$$
for some negligible function negl, *where*
  - $\mathcal{S}_1$ *is like above;*

– Q *is the list of (query, response) pairs obtained from* $\mathcal{S}_1$.

Fiat-Shamir transform. Let $(\mathsf{Prove}, \mathsf{Verify})$ be a three-round public-coin honest-verifier zero-knowledge interactive proof system (a sigma protocol) with unique responses. Let $\mathcal{H}$ be a function with range equal to the space of the verifier's coins. In the random oracle model, the proof system $(\mathsf{Prove}^{\mathcal{H}}, \mathsf{Verify}^{\mathcal{H}})$ derived from $(\mathsf{Prove}, \mathsf{Verify})$ by applying the Fiat-Shamir transform satisfies the zero-knowledge and argument of knowledge properties defined above. See Definition 1, 2 and Theorem 1, 3 in Faust et al. [20] for more details. (They actually show that these properties hold even when adversary can ask for proofs of false statements.)

B SECURITY PROOFS

B.1 Proof of Theorem 1

B.1.1 Correctness. A time bound key K_τ consists of $\eta + 1$ pairs $(\tau_1, S_1), \ldots, (\tau_{\eta+1}, S_{\eta+1})$ where $\tau_{\eta+1} = \tau$. Note that, encryption of a message for some epoch τ' is given by

$$\left(\tau, g_1^r, \mathcal{H}(\omega|_1)^r, \mathcal{H}(\omega|_2)^r, \ldots, \mathcal{H}(\omega)^r, m \cdot d\right)$$

where $d = e(\mathcal{H}(\epsilon)^r, g_1^\alpha)$. During decryption of such ciphertext using a key K_τ for which $\tau \geq \tau'$, the first task is to find the unique (τ_i, S_i) for which $\omega = M^{-1}(\tau)$ is a prefix of $\omega' = M^{-1}(\tau')$. By the doubly-labeled tree construction the uniquness is easy to see — no two nodes can have a common descendant unless one of them is an ancestor of another. By our key-generation algorithm no update key can have two such node with ancestor-descendant relationships. Now, once such pair (τ_i, S_i) is found, then assuming $\ell_i := |\omega_i|$ the decryption algorithm computes

$$\begin{aligned} & e(S_i, R) \cdot \Big(\prod_{k=1}^{\ell_i} e(\mathcal{H}(\omega_i|_k)^r, g_1^\alpha)\Big)^{-1} \\ &= \frac{e(\mathcal{H}(\epsilon)^\alpha, g_1^r) \cdot e(\mathcal{H}(\omega_i|_k)^\alpha, g_1^r) \ldots e(\mathcal{H}(\omega_i)^\alpha, g_1^r)}{e(\mathcal{H}(\omega|_1)^r, g_1^\alpha) \cdot \ldots \cdot e(\mathcal{H}(\omega)^r, g_1^\alpha)} \\ &= e(\mathcal{H}(\epsilon)^r, g_1^\alpha) = d \end{aligned}$$

The final line follows by observing that ω_i is a prefix of ω, hence $\omega_i|_k = \omega|_k$ as long as $k \leq \ell_i$ and using bilinear pairing. This conclude the proof of correctness.

B.1.2 CPA-security.

Proof. We assume a PPT adversary $\mathcal{A}$ that has a non-negligible advantage in the $\mathsf{CPA}_{\mathsf{i\text{-}TiRE},\mathcal{A}}$ game. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that attacks the decisional BDH game with non-negligible success probability. That is, $\mathcal{B}$ acts as a Game challenger to $\mathcal{A}$ (and simulates the random oracle $\mathcal{H}_1$) and uses the output of $\mathcal{A}$ to solve the following DBDH problem[10]: when given description of groups $\mathbb{G}_0$ (with generator g_0), $\mathbb{G}_1$ (with generator g_1), $\mathbb{G}_T$, bilinear map e, and values ($A_0 = g_0^a, A_1 = g_1^a, B_1 = g_1^b, C_0 = g_0^c, C_1 = g_1^c$ and $D = e(g_0, g_1)^d$), $\mathcal{B}$ must determine whether $d = abc$ or not (where $a, b, c, d \leftarrow_\$ Z_q$).

First, $\mathcal{B}$ initiates the execution of $\mathcal{A}$, who must commit to the target epoch $\tau^\star$ that it wishes to attack. Let $\omega^\star := M^{-1}(\tau^\star)$ be the primary label corresponding to $\tau^\star$ in the doubly-labeled tree Γ and also let $\ell := |\omega^\star|$ bits.

[10]we use the DBDH-3 assumption defined in [13] for Type 3 Pairings

Recall that for a bit string σ, $\sigma|_i$ denotes the i-bit prefix of σ, and we now let $\sigma|_{\bar{i}}$ denote the $(i-1)$-bit prefix followed by the negation of the i-th bit of σ. Below we often sample a uniform random value, denoted as $\chi_\omega \leftarrow_\$ Z_q$ for each node ω.

Now $\mathcal{B}$, on query ω, chooses $\chi_\omega \leftarrow_\$ Z_q$ and programs the random oracle $\mathcal{H}$ as follows:

$$\mathcal{H}(\omega) := \begin{cases} B_0 & \omega = \epsilon \\ g_0^{\chi_\omega}/B_0 & \omega \in \{\omega^\star|_{\bar{i}}, \omega^\star 0, \omega^\star 1\} \text{ for } i \in [\ell] \\ g_0^{\chi_\omega} & \text{otherwise} \end{cases}$$

Then it sets $lpk = A_0$, and give lpk to $\mathcal{A}$ and thereby implicitly sets $lsk_t := a_t$. Each update key (for time $\tau = M(\omega)$) contains exactly one S_τ values, which $\mathcal{B}$ computes as follows:

$$S_\tau = \prod_{j=1}^{|\omega|} A_1^{\chi_{\omega|_j}}$$

Note that the summation may comprise zero terms (if the root is the only common node between the path-based identities of ω and $\omega^\star$).

Remark 4 (Key Distribution). *We verify that keys given to $\mathcal{A}$ have the correct distribution.*

$$\begin{aligned} S_\tau &= \mathcal{H}(\epsilon)^a \cdot \prod_{j=1}^{|\omega|} \mathcal{H}(\omega|_j)^a \\ &= B_0^\alpha \cdot \left(\prod_{j=1, j\neq i}^{|\omega|} g_0^{\chi_{\omega|_j}\alpha}\right) \cdot (g_0^{\chi_{\omega|_i}}/B_0)^\alpha \\ &= \prod_{j=1}^{|\omega|} A_1^{\chi_{\omega|_j}} \end{aligned}$$

After some number of queries, $\mathcal{A}$ generates a challenge query with messages m_0 and m_1. $\mathcal{B}$ responds by sampling a random bit b and returning $(c_1, m_b \cdot D)$ where $c_1 := (C_1, C_0^{\chi_{\omega^\star|_1}}, \ldots, C_0^{\chi_{\omega^\star|_\ell}})$.

Remark 5 (Ciphertext Distribution). *We must verify that the ciphertext given to $\mathcal{A}$ follows the correct distribution.*

$$\begin{aligned} & (C_1, C_0^{\chi_{\omega^\star|_1}}, \ldots, C_0^{\chi_{\omega^\star|_\ell}}, m_b \cdot D) \\ &= (g_1^c, g_0^{\chi_{\omega^\star|_1} c}, \ldots, g_0^{\chi_{\omega^\star|_\ell} c}, m_b \cdot e(g_0, g_1)^d) \\ &= (g_1^c, \mathcal{H}(\omega^\star|_1)^c, \ldots, \mathcal{H}(\omega^\star|_\ell)^c, m_b \cdot e(g_0, g_1)^d) \end{aligned}$$

Finally, $\mathcal{A}$ responds with bit b', and $\mathcal{B}$ outputs 1 if $b = b'$ and 0 otherwise. Clearly, if the DBDH game sets $d = abc$, we get a valid encryption of m_b and then $\mathcal{B}$ has the same advantage at breaking DBDH as $\mathcal{A}$ at breaking CPA. On the other hand, if d is a random element of Z_p, then the last element is a random element of $\mathbb{G}_T$ and therefore the ciphertext is independent of b — probability that $\mathcal{B}$ outputs 1 is exactly 1/2. Therefore the probability that $\mathcal{B}$ succeeds in breaking the DBDH game is at least $|\mathsf{CCAadv}[\mathcal{A}, \mathsf{i\text{-}TiRE}] - 1/2|$. This concludes the proof.

□

B.2 Proof of the maliciously secure threshold scheme (Theorem 2)

This proof is very similar to the proofs provided in Agrawal et al. [2] and Christodorescu et al. [18] in the context of threshold symmetric-key encryptions. This is because the only changes made in the base i-TiRE construction (c.f. Figure 4) to enable threshold key-generation against up to $t-1$ malicious corruption (c.f. Figure 5) is in the algorithms PartUKGen and UKCombine in a way that is very similar to the maliciously secure constructions provided in those works (see, for example, Figure 4 of [3]). In particular, similar to those constructions, here too, on the corruption query, the reductions would send $t-1$ uniform random values $lsk_1, \ldots, lsk_{t-1}$ and thereby implicitly sets $lsk_t := a_t$, which is obtained by Lagrange interpolation in the exponent from $A_0 = g_0^a$ and $g_0^{lsk_1}, \ldots . g_0^{lsk_{t-1}}$. Furthermore, to achieve security against malicious adversary we use a NIZK proof for exponent with respect to a statement involving trapdoor commitments. Therefore, in the proof, the reduction additionally needs to produce dummy commitments as part of pp and simulated proofs when responding to the PartUKGen queries on behalf of the honest parties. This can be done by a few hybrids from the initial security game. We omit the details.

B.3 Proof of CCA-security (Theorem 3)

This proof is basically the same as the proof provided in the BTE scheme (see Theorem 2 of [9]). The only difference from the standard Fujisaki-Okamoto transform is the inclusion of the node ω (equivalently the epoch information) with the hash function to derive the randomness. This is necessary in our context because, otherwise one may maul the ciphertext by keeping everything same and just change the epoch (namely to a lower value than the target) to make a legitimate decryption query and subsequently break indistinguishability. We omit the details.