Time-Deniable Signatures

Abstract

In this work we propose *time-deniable signatures* (TDS), a new primitive that facilitates deniable authentication in protocols such as DKIM-signed email. As with traditional signatures, TDS provide strong authenticity for message content *for a sender-chosen period of time*. Once this time period has elapsed, however, time-deniable signatures can be forged by any party who receives a single signature. This forgery property ensures that signatures serve a useful authentication purpose for a period of time, while also allowing signers to plausibly disavow the creation of older signed content. Unlike past proposals for deniable authentication, TDS do not require the deployment of any persistent cryptographic infrastructure or services beyond the signing process (*e.g.,* APIs to publish secrets or author timestamp certificates.) To lay the groundwork for our constructions, we first investigate the security definitions for time-deniability, demonstrating that past definitions are insufficient (and indeed, allow for broken signature schemes.) We then propose an efficient construction of TDS based on well-studied assumptions.

1 Introduction

Many communication systems use cryptographic signatures to verify the authenticity of data sent from one party to another over untrusted networks. While cryptographic authentication is standard in end-to-end encrypted messaging systems, it is also increasingly being deployed within traditionally non-encrypted protocols such as SMTP email. Specifically in the email setting, protocols such as DKIM, DMARC and ARC [10] are routinely used to add non-repudiable digital signatures to email in transit between Mail Transfer Agents (MTAs): these signatures allow recipient spam filtering software to verify that it originates from the claimed sender.

While cryptographic authenticity is valuable for preventing spam and spoofing of email traffic, DKIM signatures have been re-purposed for purposes that may not have been anticipated by the designers of these protocols.[1] For example, news organizations routinely verify the authenticity of leaked or stolen email collections using DKIM signatures [33, 27]: this is possible because DKIM signing keys are long-lived, and the protocol's non-repudiable signatures can be verified long after email has been received and processed. To facilitate such verification, organizations such as the Associated Press and Wikileaks even publish detailed instructions and tools for verifying the authenticity of DKIM signatures in leaked and stolen email corpora. Since email signing is implemented by commercial mail providers rather than end-users, users of popular services cannot opt out. These developments have crated a technical debate around the desirability of long-term non-repudiability guarantees in widely-used protocols such as email [20], and raised questions around the value of adding cryptographic *deniability* to these systems.

The need for deniability. Cryptographic deniability is a property that allows communication participants to disavow authorship of messages, *e.g.,* in the event that they have been leaked or stolen. This feature has frequently been incorporated in *interactive* messaging protocols [7, 1, 35], which historically realize deniability through the use of interactive key exchange protocols and symmetric authentication primitives such as MACs. Achieving deniable authentication in email authentication protocols such as SMTP/DKIM is more challenging, since these protocols support non-interactive and asynchronous delivery via multiple

[1] Indeed, many early deployments of DKIM used weak signing keys, and some DKIM standards authors proposed using *e.g.,* 600-bit keys to balance the risks and benefits of DKIM [10].

intermediate recipients. Thus interactive protocols are ruled out, and even designated-verifier solutions can be more challenging due to the presence of intermediaries.

Despite these challenges, the problem of incorporating deniability for the email setting has recently received some attention. For example, in Usenix Security 2021, Specter *et al.* proposed two technical replacements for DKIM signing that are designed to facilitate deniability. Both protocols ensure that messages are digitally signed to enable sender-authenticity verification, but feature a process wherein senders, recipients and even third parties parties can create deliberate *forgeries* after the necessary anti-spam and spoofing checks have been completed. These protocols employ two techniques: the first relies on the sender to author forgeries on request and/or publish expired secret keys, while the second employs a trusted *time server* that publishes cryptographic timestamp certificates that allow forgery of signatures after some period of time has elapsed. Others have made even simpler proposals wherein DKIM providers simply rotate and publish existing DKIM signing keys on a periodic basis [20]. Each proposal seeks to build signatures that are unforgeable for a period of time necessary to support short-term transport checks, but allows forgery at a later point with the cooperation of the sender, recipient or some other infrastructure.

The major limitation of the proposals above is that forgery requires the active cooperation of signers, or else depends on the creation of new trusted infrastructure such as "time servers" that publish keys or timestamp certificates on a periodic basis [34]. The challenge in email systems is that the end-users affected by non-repudiable authentication (*e.g.,* Gmail customers) rely on third-party providers to deploy these infrastructure services and make them available for the controversial purpose of forging past email. If this infrastructure is not deployed, then even the Internet-wide adoption of a deniable signature standard will not provide deniability *in practice.* What is needed is a signature scheme that can be used in place of a normal signature scheme within protocols; provides strong authenticity for a period of time; and then subsequently becomes plausibly forgeable *by any party who simply obtains such a signature*, provided that parties have an (approximately) shared view of time. We refer to such signatures as *time-deniable signatures.*

Properties of time-deniable signatures. Time-deniable signatures operate much like a normal signature scheme, but with some important differences. Like standard digital signatures, time-deniable signatures are designed to be secure and non-repudiable for at least some time period following signing. The duration of this time period is strictly limited, however: any party who obtains a signature on some message M can use it as input to a new forging algorithm called $\mathsf{AltSign}$ that, after enforcing some approximate time delay, will output a forgery on a new chosen message M'. A key requirement of these schemes is that neither signing nor forging should require the cooperation of any other party or infrastructure. This time delay is therefore enforced using a specific computational assumption: the $\mathsf{AltSign}$ algorithm requires the forger to perform a pre-specified number of *sequential* operations δ, and the minimum time required for this calculation is roughly as long as the desired length of the unforgeable phase.

Of course, the ability to forge signatures has no bearing on deniability if the resulting forgeries are easily distinguishable from authentic signatures. To achieve plausible deniability, we require that the resulting forgeries be indistinguishable from signatures produced using the ordinary signing algorithm, and in fact that forgeries should not even be *linkable* to the original signatures that were used to create them. This simple property has important follow-on implications: since forgeries are indistinguishable from true signatures, this implies that forgeries can be used to create still further forgeries.

Finally, we wish time deniable signatures to be useful in practice. Given the description above, time-deniable signatures would be of limited usefulness: the revelation of a single signature would allow for an unlimited number of forgeries, rendering the signing key useless for authenticating further messages. To remove this limitation, we slightly relax our forgery and unlinkability requirements. Our constructions allow for renewability via an additional *timestamp* t field that is specified in the signing algorithm and carried with the signature. Forgers can produce a new signature on a message M' provided the new message carries a new timestamp $t' \leq t$. For example, in a practical deployment, the timestamp t can be set to correspond to some real-world time counter, and recipients can choose to accept as authentic any signature with timestamp greater than $t - t_\delta$ where t_δ is the minimal expected time needed to compute a forgery.[2]

[2]This naturally relaxes the unlinkability requirement: given a pair of signatures $\sigma_{t_1}, \sigma_{t_2}$ with timestamps $t_1 < t_2$ it cannot be the case that σ_{t_1} is the original signature and σ_{t_2} is the forgery. However given a sufficiently large collection of signatures

Our contributions. In this work we investigate the problem of building time-deniable signatures. We first develop formal definitions for this new primitive, and then present a construction that is based on several efficient components. Finally, we implement our approach and show that it is practical enough to deploy today. Concretely, we provide the following contributions:

Defining time-deniable signatures (TDS). We propose new definitions for the concept of time-deniable signatures, and propose strong security definitions for this new primitive. Defining security for time-deniable signatures is surprisingly difficult: in the course of developing our definitions, we found that previous efforts to formalize the security of deniable authentication schemes falls short. For example, we show that the security definitions for some related primitives [21] contain subtle weaknesses that admit practically-insecure constructions. To provide evidence for the correctness of our definitions, we prove that our definitions are strictly stronger than these earlier definitions.

Efficient constructions. To demonstrate that the TDS primitive is practical, we propose an efficient construction of time-deniable signatures based on well-studied cryptographic assumptions. Our constructions improve on previous work [21] in that they do *not* require any *a priori* bound on the number of time epochs that the scheme can handle. We also show that TDS does not require the use of complex non-black box proof techniques, and can in fact be realized using standard assumptions in pairing-based cryptography as well as sequential puzzles based on repeated-squaring assumption [31].

Implementation and performance experiments. To further motivate the usefulness of TDS in systems applications, we implement our TDS constructions and show that the scheme has practical runtime and bandwidth performance than previous deniable authentication systems. In particular, we show that our scheme has a fast key setup time, which is particularly important for a scheme with an unbounded number of time epochs.

2 Technical Overview

We now give an overview of the main contributions in this work, starting with formalizing the notion, before moving on to the constructions.

2.1 Defining Time-Deniable Signatures

We study signature schemes where signatures remain valid for a short period of time after creation. Specifically, we consider the notion of an *unforgeability period* that starts when a signer generates a signature for a message using its signing key sk, and the signing algorithm Sign. But once the *unforgeability period* elapses, any participant in the system can compute a "fake signature" (aka *forgery*). To allow computation of forgeries, we consider an alternate signing algorithm $\mathsf{AltSign}$, that *does not require* the signing key sk to generate signatures. Intuitively, as long as the signatures generated by Sign and $\mathsf{AltSign}$ appear indistinguishable, such a notion provides *deniability* after the unforgeability period since a signer can claim that a signature attributed to them could have been generated by anyone.

Key Challenges in the Definition. There are several key considerations towards formalizing the above intuition and defining time-deniable signatures.

Challenge I: Preventing pre-compution of forgeries. Recall that any party can compute a forgery (via the algorithm $\mathsf{AltSign}$) after the *unforgeability period* expires. But how do we ensure that a party cannot execute $\mathsf{AltSign}$ *in advance*, thereby having the ability to sign any message *within* the unforgeability period?

One natural approach is to bind signatures to some unpredictable *cryptographic beacon*, perhaps generated at regular intervals by a centralized server or a blockchain [14, 29].[3] For example, when signing a message

containing forgeries and original signatures, this approach still provides a degree of uncertainty for all but the most recent signature.

[3]This approach was indeed proposed in a recent unpublished work by Bonneau *et al.* [2].

m (via Sign or AltSign) one might actually sign the pair (m, b) where b is a beacon released at a time known by the receiver. This value b can then be used as the "seed" to allow forgery using AltSign, and verifiers can use the known publication time of b to determine whether the signature is still within the unforgeability period. Such models have been considered in prior works, including the scheme of Specter *et al.* [34] and a recent proposal by Bonneau *et al.* [2].

In this work, we seek to avoid the use of unpredictable timestamps or centralized servers. In our notion, the Sign and AltSign algorithms do indeed take as input a timestamp t. Assuming that receivers possess (loosely) synchronized clocks, these timestamps can be used to verify that a received signature is within the unforgeability period. However, crucially, these timestamps are simply the output of a predictable clock operated by the signer, which means that we *do not require any security properties* of this input, nor do we require unpredictable beacons or new infrastructure to produce them. To prevent pre-computation, we instead model AltSign such that it requires a *valid signature* on some pair (m, t) as input. This ensures that forgers do not have the necessary input(s) to pre-compute forgeries until they obtain a signature.[4]

Challenge II: Selecting forged timestamps. In the proposal above, AltSign requires a valid signature on some time t (and any message) in order to compute a forgery. Naturally the resulting forgery will also need to contain its own timestamp t'. The selection of t' is crucial, however: if this forged timestamp can be chosen by the forger, then an attacker may be able to forge new signatures that appear (to an honest receiver) to be within the unforgeability window, even when the original signature was not. One obvious solution to this problem is to restrict the forged timestamp to $t' = t$. Unfortunately, this restriction weakens the deniability properties of the signature scheme: a signer can deny having signed a particular message at time t, but it *cannot* deny having signed *some* message at time t. To achieve stronger deniability where a signer can also deny having signed *any* message at time t, we further strengthen the AltSign algorithm. Namely, we require that on input a signature on timestamp t, AltSign can compute forgeries for any message m and *any* time stamp $t' \leq t$.

Challenge III: Avoiding clock synchronization. The closely related prior work of *epochal signatures* by Hülsing and Weber [21] consider a security notion that crucially relies on various participants having synchronized clocks. Roughly, in a epochal signature scheme, (real) time is divided into discrete *epochs* where a new key is generated at the start of every epoch. Signatures are associated with the epoch they were generated in, where unforgeability requirements states that no adversary can forge signatures for an epoch during the epoch. As we show in §3, any epochal signature scheme can be transformed into another epochal signature scheme that becomes *completely insecure when the clocks are slightly out of sync.*

Unfortunately, achieving clock synchronization is not easy, and in this work we seek to avoid such issues. Our definitions model time by the number of sequential computation steps [31, 5, 36, 30, 13], which this still requires conversion when used in the real world. But more importantly, our definition allows the adversary to participate in a "pre-processing" phase to ensure the robustness of our notion in scenarios where there may be clock synchronization issues.

Our Definition. We are now ready to provide an (informal) definition of time-deniable signatures. We refer the reader to the technical sections for more details.

The protocol is parameterized by Δ, the duration of the *unforgeability period*, and described by the algorithms Gen, Sign, AltSign and Verify. The Gen and Verify algorithms are the same as standard signature schemes while the Sign algorithm, also similar to the standard notion, takes in as input a message m and time stamp t to generate a signature on (m, t). The main new component is the algorithm AltSign which takes as input a message m', time stamp t', signature $\sigma_{(m,t)}$ such that $t' \leq t$, and uses the verification key to generate a signature $\sigma_{(m',t')}$.

For correctness of the scheme, we require that AltSign generates a verifying signature as long as its given as input the output of the Sign algorithm, or (repeated applications) of the AltSign algorithm. We now provide an overview of the two key security properties required by our notion.

[4]Indeed, we show that the need for AltSign to use an existing signature (or portion thereof) to produce a forgery is seemingly inherent if we do not want to use secure infrastructure. We elaborate on this point in Appendix F.

Unforgeability. This property captures the notion that no adversary capable of computing fewer than Δ sequential steps can generate a forgery. Specifically, we allow an initial *pre-processing stage* for the adversary where it is *not* bounded by the number of sequential steps, gathering as much information as it can. At the end of this stage, say at timestamp t^*, it passes along any information onto the next stage where the adversary that runs in at most Δ sequential steps needs to produce a signature for a message with time stamp $> t^*$.

Deniability. This property asks an adversary to distinguish between a "fresh" signature generated using Sign, and a signature generated using $\mathsf{AltSign}$. We formalize this by defining two experiments, where the adversary is allowed to specify a tuple $(m_1, t_1, \sigma_1 = \mathsf{Sign}(m_1, t_1), m_2, t_2)$ with $t_2 \leq t_1$. In the first world, the output is simply the signature $\sigma_2 = \mathsf{Sign}(m_2, t_2, \mathsf{sk})$, whereas in the second world the output is $\sigma_2 = \mathsf{AltSign}(m_2, t_2, \sigma_1, \mathsf{vk})$. We say a TDS is deniable if no computationally bounded adversary can distinguish the two with a significant probability.

We refer to the above description of deniability to be "1-hop-deniable", i.e. a signature generated via Sign is indistinguishable from one generated via $\mathsf{AltSign}$. In the technical section, we extend this notion to "k-hop-deniability", which intuitively corresponds to the indistinguishability between a signature generated via Sign and one generated via k *applications of* $\mathsf{AltSign}$.

2.2 Construction

Time-Deniable Signatures from Delegatable Functional Signatures. Our construction centers around the following natural idea: with each signature produced by the signer, we leak a *restricted signing oracle* that can be used to forge later signatures. A signing oracle, as the name suggests, allows a party with access to the aforementioned oracle to sign any message of its choice. For instance, the signing key can be viewed as an oracle since it allows one to sign any message of their choice. A restricted signing oracle limits the messages that can be signed. Thus, continuing with our analogy of signing keys corresponding to an oracle, a restricted signing oracle corresponds to a signing key that is restricted in a fine-grained manner.

When the Sign algorithm generates a signature on message m and time stamp t, it also reveals a restricted signing key sk_t that can be used sign any message m' with time stamp $t' \leq t$. Such a key can then be used by the $\mathsf{AltSign}$ algorithm to create forgeries. Revealing the restricted key with the signature, however, allows anyone in possession of the signature to create forgeries *during* the unforgeability period. To prevent this, we need to *hide* this restricted signing key until after the unforgeability period, and we do so using *time-lock puzzles*. Intuitively, a time-lock puzzle allows one to "lock" a secret s for a predetermined amount of time (i.e., time parameter). Thus, the output of the Sign algorithm will consist of the signature $\sigma_{m||t}$ along with the time-lock puzzle containing the secret sk_t, computed with time parameter Δ. We note that a similar approach has been considered in constructing notions such as epochal signatures [21], and we refer the reader to Section 3 for a more detailed comparison.

To implement restricted signing keys, we turn to the notion of *functional signatures* (FS) [8, 4, 3]. Functional signatures are equipped with *functional keys* sk_f (instead of "regular" signing keys) such that it allows one to sign $f(m)$ for any message m. We consider the following specific function for our application:

$$f_T(t, m) = \left\{ \begin{array}{ll} t||m & t \leq T \\ \perp & \text{otherwise} \end{array} \right\} \tag{1}$$

We call such functions *prefix functions* (the function prepends the time stamp to the message). It is evident from the above description that with a functional key sk_{f_T} one can generate a signature for any message m and time stamp t as long as $t \leq T$.[5]

For our TDS construction, we leverage specific properties of the functional signature scheme. We provide a more general (and detailed) definition in the technical sections, but for the purposes of the overview, we shall discuss the relevant properties of functional signatures for the specific function f_T described above: (i)

[5]Note that while we have thus far described signatures on messages of the form $m||t$, the above description of f_T flips it to be $t||m$. Looking ahead, the change is due to our construction of functional signatures.

delegatability: given a key sk_{f_T} for function f_T, using only public parameters, one can derive a key $\mathsf{sk}_{f_{T'}}$ for a function $f_{T'}$ if $T \leq T'$; (ii) *key indistinguishability*: it should be computationally infeasible to differentiate between a fresh key sk_{f_T} and a key derived ; and (iii) *unforgeability*: it should be computationally infeasible to generate signatures $\sigma_{m||t}$ unless one has the key sk_{f_T} where $T \geq t$. While delegatability has previously been studied for functional signatures, the notion of key indistinguishability is new to our work. The latter is crucial towards achieving deniability.

Putting things together, we have:

Sign On input message m and time stamp t, the Sign algorithm generates the key sk_{f_t} (using the *master secret key*, see technical section for details), and uses it to compute the signature $\sigma_{t||m}$. It then encrypts the key sk_{f_t} within a time-lock puzzle with time parameter Δ.

AltSign On input message m', time stamp t', and signature $\sigma_{t||m}||\mathsf{TimeLock}(\mathsf{sk}_{f_t})$, the AltSign algorithm first solves the time-lock puzzle to obtain sk_{f_t}. It next uses the delegation functionality to derive a key $\mathsf{sk}_{f_{t'}}$ from sk_{f_t} and then follows the description of the Sign algorithm.

A useful property of the above approach is that the *sequential* part of the computation performed by AltSign, namely, solving the time-lock puzzle, can be *reused for computing many forgeries in parallel*. This is because once the restricted signing key is obtained – a one-time work, it can be used to compute signatures in parallel.

Intuitively, we prove unforgeability by leveraging the unforgeability of the functional signature scheme and the security of time-lock puzzle, while deniability follows from the key indistinguishability property of the functional signature scheme.

Prefix Function FS from Hierarchical Identity Based Encryption. We construct functional signatures for prefix functions from Hierarchical Identity Based Encryption (HIBE). At a high level, HIBE is an encryption scheme that allows one to encrypt to identities, (treated as bit strings in this work) such that only someone in possession of the secret key corresponding to the identity can decrypt messages. The hierarchical nature of the scheme allows for the delegation of keys, i.e. if one is in possession of a key for identity $\mathcal{I}$ which is a prefix of an identity $\mathcal{I}'$, one can derive the key for $\mathcal{I}'$ from the key for $\mathcal{I}$. The identities in our setting will correspond to the nodes of a binary tree with nodes labeled by binary strings corresponding to their path from the root (left is 0, right is 1).

HIBE schemes can be used generically to construct a signature scheme [6] - to sign a message m, use the HIBE scheme to generate a key for the "identity" m with the key corresponding to the signature. The verification of the signature is performed by encrypting a random message to the message (treated as the identity), and using the signature as a key to check whether the decryption is correct.

In our setting, the identities will be the bit strings corresponding to $t||m$. Structuring as above has the following benefit - if one were in possession of a HIBE key for a time stamp t, then one can derive keys for $t||m$ for *any* message m since $t||m$ is "lower" in the hierarchy from t. Therefore to sign a message m at time stamp *exactly* t it suffices to possess the key for t, which serves as the signing key. But recall from the description of f_t in the prior section, the signing key corresponding to f_t should allow one to sign messages for *any* time stamp smaller than t. A naive way would be to generate the signing key for f_t would be to concatenate the HIBE keys for all $t' \leq t$, but this is approach is clearly infeasible since the signing key would grow linearly with the total number of possible time stamps.

To overcome this efficiency barrier, we leverage the tree structure of the HIBE scheme with the following insight - it suffices to have a small number of keys as long as we are able to derive keys for any $t' \leq t$. At a high level, the signing key sk_{f_t} will consist of keys for all identities that are the *left siblings* of the nodes along the path from $t+1$ to the root[6], resulting in at most $\log(t)$ many keys. A detailed description is provided in the technical sections, but here we provide an illustrative example.

From Figure 1, it is clear that to derive the key for $\mathsf{sk}_{f_{t'}}$ from sk_{f_t} one can simply use the HIBE delegation algorithm, i.e. there is no need to run the key generation algorithm afresh.

[6]One can also view it as the nodes in the stack during the depth first traversal of the (identity) binary tree when node $t+1$ is visited.

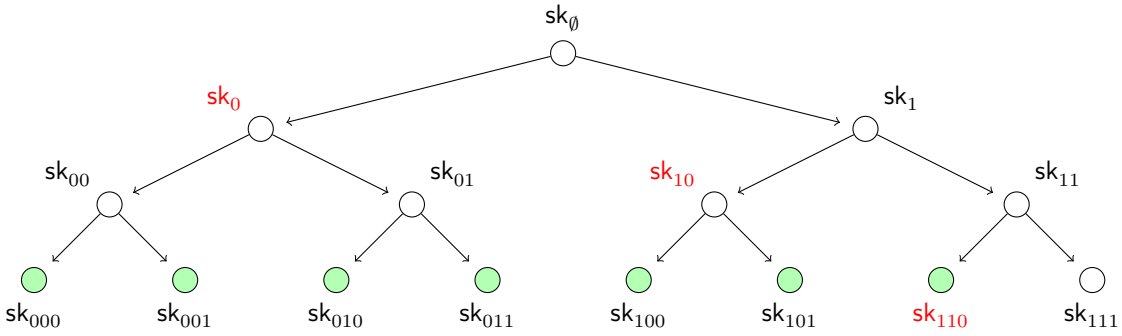

Figure 1: This tree represents the hierarchical nature of the identities in our HIBE scheme. Each node in the tree besides the root represents a HIBE secret key $\mathsf{sk}_{\mathsf{id}}$ for the identity id. Therefore, by our description $\mathsf{Trace}(root = \emptyset, 110)$ will constitute the nodes which represent the secret key for f_{110}, i.e. $\mathsf{sk}_{f_{110}} = (\mathsf{sk}_0, \mathsf{sk}_{10}, \mathsf{sk}_{110})$. Using this set, all messages with prefixes in the green nodes can be signed.

Looking ahead, since we want to allow the adversary to choose the message it wants to compute a forgery on *after* it has seen other signatures, we require the HIBE scheme to be *adaptively* secure (i.e. the adversary can choose the identity of the HIBE scheme it wants to break *after* seeing keys for other identities). HIBE schemes satisfying the necessary requirements can be instantiated e.g., assuming the Decisional Linear (DLIN) assumption on Bilinear groups of prime order [23].

3 Related Work

Concurrent work. A concurrent and independent work of Arun et al. [2] also studies a notion similar to time-deniable signatures. Similar to our work, they make use of sequentially-ordered computation as a means to enforce time delay during which signatures are unforgeable, but become forgeable afterwards. However their work considers a different models than ours. Specifically, their system relies on the use of *unpredictable beacons* that are presumably released periodically by some trusted outside source. In contrast, we do not do rely on any randomness beacons or time servers. Unlike our work, they also explore time-based deniability in proof systems.

Epochal Signatures. Our work is closely related to the prior work on epochal signatures [21]. At a very high level, epochal signatures aim to achieve deniability in a manner similar to our - by leaking a constrained key. In epochal signatures, (real) time is partitioned into discrete *epochs* with a key update mechanism at the start of every epoch. Any signature generated during epoch i additionally include the keys for prior epochs, allowing for forgery of signatures of any epoch $< i$ (but not epoch i).[7] The constructed epochal signature in [21] leaks only a *single* key with the property that from a key of epoch i, k_i, one can retrieve the key of epoch $i - j$, k_{i-j} with j applications of a "key retrieval" function, but security requires that it is impossible to retrieve keys for epochs $> i$ from k_i.

In the following, we describe some key differences between the two works.

Bounded vs Unbounded Use. Unlike our work, the system proposed in [21] is limited to be *bounded use*. Specifically, the number of epochs that their system can support is bounded ahead of time. This is an outcome of the running time of their system setup, which is linear in the number of epochs.

In practice, the granularity of each epoch and the number of epochs must be fixed *before* their system is initialized, and once the total number of epoch surpass the bound, the entire system needs to be reset from

[7] In their work, they consider an additional *deniability* parameter V such that signatures for epoch i include keys for epochs $i - V$ and earlier allowing for V epochs where the signature is valid. But for the purposes of this discussion we describe it in the above simplified manner.

scratch, a scenario that is clearly undesirable. For epochs that are sufficiently small, such a restriction would *limit the total number of signatures* that the system can support. Indeed, for similar reasons, the broad question of *bounded* vs *unbounded* use has been studied in various contexts in cryptography such as bounded vs unbounded query chosen-ciphertext secure encryption [11], depth-bounded vs depth-unbounded hierarchical identity-based encryption [24] and homomorphic encryption [16], bounded-collusion vs unbounded collusion in functional encryption [32, 19], and more. In all of these cases, there are significant challenges and overheads (in terms of assumptions, efficiency, etc) in going from bounded system to an unbounded one. As such, we view our improvements over [21] in this regard to be a significant one.

Need for Clock Synchronization. The unforgeability notion in [21] deems an adversary to be a valid forger if it can successfully forge a signature *within* an epoch e after observing signatures associated with that epoch. This immediately puts a requirement on clock synchronization across all participants, which is often quite difficult to achieve in practice.

We now demonstrate that if such a requirement is not met, then the consequences can be catastrophic and result in a compromise of security. Specifically, we construct a secure epochal signature scheme where the unforgeability property can be broken when the clocks are slightly non-synchronized. We also show that the same scheme – translated to the setting of time-deniable signatures – is *not* secure as per our definition, thus demonstrating that the latter is a strictly stronger notion.

In the following, we give a simplified presentation of the counter-example to convey the general idea. The full counter-example is more involved (due to technical reasons) and is presented in Appendix G.1.

Intuitively, we exploit the fact that in the unforgeability definition of epochal signatures, the adversary can only query signatures for an epoch e during epoch e. We then associate with each epoch e, a special *trigger* message m_e^* such that if the adversary queries for a signature on message m_e^* *during* epoch e, then it receives some "secret information" from the signing oracle. Let us start with any secure epochal signature scheme with epoch length Δt. We modify the scheme such that during epoch e, the new scheme also outputs (i) a time-lock puzzle encrypting m_e^* with difficulty parameter Δ_t' such that $\Delta t < \Delta t' < \Delta t + \varepsilon$; (ii) the secret signing key sk XORed with a random string r. The difficulty parameter of the time-lock puzzle ensures that the puzzle *cannot* be decrypted within the epoch that it is generated, but can be decrypted just after the epoch concludes. Thus, if there is a clock synchronization issue where the challenger's (the entity generating the signatures) clock is slightly slower, then an adversary can decrypt to obtain m_e^* and query the signing oracle on m_e^* to obtain the "secret information" - which is the random string r. This in turn can be used to obtain sk and forge any signature in the future, breaking the unforgeability in this scenario.

To argue that this scheme is a secure epochal signature scheme when the clocks are synchronized, we note that in an epochal signature scheme, at the start of an epoch $e+1$ two things happen: (i) key evolution procedure is applied to the secret signing key to generate the signing key for the next epoch; and (ii) public information pinfo_e is broadcast. Here, pinfo_e allows anyone to produce signatures for epochs $\leq e$ without the signing key such that they are indistinguishable from signatures produced by the real signing key (akin to our definition of deniability). In the above scheme, while the secret key consists of the random string r, this is not revealed as a part of pinfo_e. Thus, simply having pinfo_e is not good enough to retrieve sk since it does not contain the random string r needed to do so.

We now argue that the above scheme is *not* a secure time-deniable scheme. Briefly, this is due to the pre-processing phase we allow during the unforeability definition. In this phase, the adversary can query the same time stamp multiple times (here roughly the time-stamps correspond to an epoch), and therefore can perform the above described attack by decrypting the time-lock puzzle and using it to query the same time stamp on m_e^*, thereby obtaining the signing key that is passed to the "online adversary" that simply uses the signing key to produce a forged signature.

We remark that the above description is oversimplified and the full counter-example is presented in G.1.

4 Preliminaries

Throughout the paper we consider the depth $\mathsf{depth}(C)$ of a circuit C to be defined as the longest path in the circuit from input wires to output wire. The size of a circuit $\mathsf{size}(C)$ corresponds to the number of gates.

Sequential time. In this work sequential time refers to the non-parallelizable time it would take any circuit to compute a particular function. More formally, a function has sequential time d or takes d sequential steps if there exists a circuit C which correctly computes that function and the depth $\mathsf{depth}(C) = d$. This notion attempts to capture inherent limitations in computing a function that cannot be overcome by access to more cores or processors.

Time-lock Puzzles. The concept of a time lock puzzle or time lock encryption was first introduced by Rivest, Shamir, and Wagner [31]. In this definition, a solution is provided as *input* to an algorithm which produces puzzles as output. Security requires that it is hard to figure out what solution was hidden inside the puzzle unless someone is willing to spend sequential time proportionate to a function of t, if t is a time parameter for our scheme. We use the circuit-based adversarial definition which translates to bounding the parallel time of the adversary via bounding the adversary's depth.

Definition 4.1. A puzzle $\mathsf{TimeLock}$ is a tuple of algorithms $\mathsf{Gen}, \mathsf{Sol}$ where the signature of the algorithms is defined as below.

$\mathsf{Gen}(1^\lambda, \Delta, s) \to Z$: On input a time/difficulty parameter Δ and a solution $s \in \{0,1\}^\lambda$, output a puzzle Z

$\mathsf{Sol}(\Delta, Z) \to s$: This is a deterministic algorithm that when given a puzzle Z and the difficulty parameter Δ produces a solution s.

Correctness. Correctness requires that for all solutions $s \in \{0,1\}^\lambda$ and difficulty parameters Δ the following holds:

$$\Pr\left[Z \leftarrow \mathsf{Gen}(1^\lambda, \Delta, s) : s \leftarrow \mathsf{Sol}(\Delta, Z)\right] = 1$$

Efficiency. $\exists$ a polynomial p s.t. $\forall \Delta, \lambda \in \mathbb{N}$, $\mathsf{Sol}(\Delta, \cdot)$ runs in time $\Delta \cdot p(\lambda)$

Security. We consider a time lock puzzle to be α-gap secure if $\forall$ functions $T(\lambda) \geq \alpha(\lambda)$ and distinguishers $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of size $\mathrm{size}(\mathcal{A}_\lambda) \in \mathrm{poly}(\lambda)$ and depth $\mathrm{depth}(\mathcal{A}_\lambda) \leq \frac{T(\lambda)}{\alpha(\lambda)}$, $\exists$ a negligible function μ s.t.$\forall \lambda \in \mathbb{N}$, $\forall s_0, s_1 \in \{0,1\}^\lambda$,

$$|\Pr\left[Z \leftarrow \mathsf{Gen}(1^\lambda, T(\lambda), s_0) : \mathcal{A}_\lambda(Z) = 1\right] - \Pr\left[Z \leftarrow \mathsf{Gen}(1^\lambda, T(\lambda), s_1) : \mathcal{A}_\lambda(Z) = 1\right]| \leq \mu(\lambda) \qquad (2)$$

This is a variation of the time lock puzzle definition of [12], where we define security to hold for adversaries of polynomial size instead of super polynomial.

4.1 Hierarchical Identity Based Encryption

We recall the notion of Hierarchical Identity Based Encryption (HIBE). A HIBE scheme has the following five algorithms:

$\mathsf{Setup}(1^\lambda) \to (msk, pk)$: The setup algorithm generates the master secret key and public parameters.

$\mathsf{KeyGen}(msk, I) \to sk_I$ Generates a key for the identity I using the master secret key msk.

$\mathsf{Delegate}(pk, sk_I, I') \to sk_{I||I'}$: Takes a secret key of some identity I and generates a secret key for the identity $I||I'$.

$\mathsf{Encrypt}(pk, m, I) \to ct$ The encryption algorithm takes the public key, a message and an identity I and outputs the corresponding ciphertext.

$\mathsf{Decrypt}(sk_{I'}, ct) \to m/\perp$: The decryption algorithm takes a secret key and a message and outputs the message if the secret key hierarchy level allows decryption of the ciphertext.

Remark. Throughout this paper, we will make use of HIBE schemes where $\mathsf{Delegate}$ can take in a child identity I' that is the empty string. In such schemes, $sk'_I \leftarrow \mathsf{Delegate}(pk, sk_I, nil)$ is a re-randomization of the key sk_I for identity I. We note that the HIBE scheme by Lewko-Waters[24] and its variants[23] can be modified to have this property.

4.1.1 Security

We consider HIBE security similarly to the work of Lewko and Waters[24, 25] using the following security game played by a challenger and an adversary.

- **Setup** The challenger runs $(pk, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives the public parameters pk to the adversary. Let set $\mathcal{S}$ be the set of private keys that the challenger creates. At the beginning, $\mathcal{S} = \emptyset$.
- **Phase 1** In this phase, the adversary gets to make three types of queries.
  1. Create queries $\mathcal{QC}(I)$, which are made on some specific identity I. The challenger adds the keys for this identity to the set $\mathcal{S}$. Note that the adversary does not get these keys.
  2. Delegate queries $\mathcal{QD}(I)$, which are made on some identity I such that the corresponding keys are in the set $\mathcal{S}$. The challenger adds the keys corresponding to the delegated identity I' and adds them to the set $\mathcal{S}$.
  3. Reveal queries $\mathcal{QR}(I)$, which are also made on some identity I such that the corresponding keys are in the set $\mathcal{S}$. In response, the challenger gives the corresponding keys to the adversary and removes them from the set $\mathcal{S}$.
- **Challenge** The adversary gives the challenger messages m_0 and m_1 and a challenge identity I^*. The challenger responds with a random $\beta \in \{0, 1\}$ and encrypts m_β under I^* and sends the ciphertext to the adversary.
- **Phase 2** The adversary gets to query the challenger similar to Phase 1.
- **Guess** The adversary outputs its guess β' for β and wins if the following conditions are satisfied:
  1. $\beta' = \beta$.
  2. The challenge identity I^* should satisfy the property that no revealed keys, in either of the query phases, belong to an identity that was a parent of I^* and the I^*'s keys shouldn't have been revealed.

  The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = \Pr[\beta' = \beta] - \frac{1}{2}$.

Definition 4.2 (Adaptive security for HIBE)**.** A HIBE scheme is adaptively-secure if $\forall$ poly size adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ their advantage $\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ in the HIBE security game defined above is negligible.

4.1.2 Key-indistinguishability

We additionally require that all polynomial-time adversaries have at most a negligible advantage in distinguishing between keys generated via the KeyGen algorithm and keys generated via the $\mathsf{Delegate}$ algorithm even when given access to the master secret key msk. We define this property using the following HIBE key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{HIBE}}$.

The **Setup** phase is similar to the HIBE security game above, except the adversary now also gets the master secret key msk. Similarly, the set $\mathcal{S}$ of keys queried and the corresponding query identifier is set to be empty. A bit $\beta \xleftarrow{\$} \{0, 1\}$ is sampled uniformly.

Query phase. In this phase the adversary is allowed to adaptively query a key oracle $\mathcal{QK}(\cdot)$ and a challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot, \cdot, \cdot)$

$\mathcal{QK}(\cdot)$ takes as input an identity I, computes $sk_I \leftarrow \mathsf{KeyGen}(msk, I)$, selects an identifier id and adds (id, I, sk_I) to the set $\mathcal{S}$ and responds with (id, sk_I).

$\mathsf{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ takes as input a challenge identifier id, and a pair of identities (I_0, I_1) such that I_0 is a *parent identity* of I_1. Where parent identity implies that on the hierarchical identity tree (Figure 1) where the root is msk, I_0 is an intermediate node on the shortest path from I_1 to the root. It checks for id in set $\mathcal{S}$ and checks that id corresponds to a key query on I_0. If no such id is found, the output is $\perp$. Otherwise, compute $sk_0 \leftarrow \mathsf{KeyGen}(msk, I_1)$, $sk_1 \leftarrow \mathsf{Delegate}(pk, sk_{I_0}, I_1)$ and respond with sk_β.

Guess. The adversary outputs its guess β' for β and wins if $\beta' = \beta$.

The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = \Pr[\beta' = \beta] - \frac{1}{2}$.

Definition 4.3 (Key-Indistinguishability for HIBE)**.** A HIBE scheme is delegated key indistinguishable if $\forall$ poly size adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ their advantage $\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ in the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{HIBE}}$ game is negligible.

The key indistinguishability property is satisfied by many existing HIBE schemes in the literature. For completeness, we prove in Appendix D that it satisfied by Lewko's HIBE scheme [23].

5 Time-Deniable Signatures: Definition

A time-deniable signature scheme $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{AltSign})$ is a tuple of possibly probabilistic polynomial-time algorithms:

$\mathsf{KeyGen}(1^\lambda, T = T(\lambda)) \rightarrow (vk, sk)$: On input the security parameter 1^λ and a difficulty parameter for $\mathsf{AltSign}$ called T, this randomized algorithm outputs the verification key vk and the signing key sk.

$\mathsf{Sign}(sk, m, t) \rightarrow \sigma$: On input a message m and the signing key sk, this randomized algorithm outputs a signature σ on m for timestamp t.

$\mathsf{Verify}(vk, \sigma, m, t) \rightarrow \{0, 1\}$: On input a signature σ, a message m, verification key vk and timestamp t, this deterministic algorithm outputs a bit.

$\mathsf{AltSign}\,(vk, (m^*, t^*, \sigma^*), m, t) \rightarrow \sigma$: On input a valid message and signature pair (m^*, σ^*) for timestamp t^*, this randomized algorithm outputs a signature σ on message m for timestamp t.

Definition 5.1 (Efficiency)**.** The algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}$ must run in time poly in the size of the input. For $\mathsf{AltSign}$ it is required that there exist a positive polynomial q such that $\forall T = T(\lambda), \forall\lambda \in \mathbb{N}$, $\mathsf{AltSign}$ is computable in time $q(\lambda) \cdot T$ where T is the difficulty parameter provided to KeyGen.

Definition 5.2 (Correctness)**.** A time-deniable signature scheme for a message space $\mathcal{M}$ satisfies the correctness property if it satisfies the following two conditions:

1. $\forall m \in \mathcal{M}, (vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda), \sigma \leftarrow \mathsf{Sign}(sk, m, t)$, it holds that $\mathsf{Verify}(vk, \sigma, m, t) = 1$.
2. Let $\mathsf{AltSign}^k(vk, (m_0, t_0, \sigma_0), \{(m_j, t_j)\}_{j\in[k]})$ be shorthand for the following recursively defined function:

$$\begin{aligned}\mathsf{AltSign}^i(vk, (m_0, t_0, \sigma_0), \{(m_j, t_j)\}_{j\in[i]}) = \\ \mathsf{AltSign}(vk, (m_{i-1}, t_{i-1}, \mathsf{AltSign}^{i-1}(vk, (m_0, t_0, \sigma_0), \\ \{(m_j, t_j)\}_{j\in[i-1]}), m_i, t_i)\end{aligned}$$

where $\mathsf{AltSign}^0(vk, (m_0, t_0, \sigma_0), \{\}) = \sigma_0$. In words, $\mathsf{AltSign}^k$ is a signature obtained by applying $\mathsf{AltSign}$ k times to a provided signature σ_0 on the message m_0, t_0. Then we have the following additional correctness property:

$\forall k \in \mathbb{N}$, for all sets of ordered tuples $\{(m_j, t_j)\}_{j\in[k]}$, and $\forall m_0, t_0$ that satisfy $t_{j-1} \geq t_j$ where $j \in [k]$:

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T);\\ \sigma_0 \leftarrow \mathsf{Sign}(sk, m_0, t_0);\\ \sigma \leftarrow \mathsf{AltSign}^k(vk, (m_0, t_0, \sigma_0),\\ \{(m_j, t_j)\}_{j\in[k]}) :\\ \mathsf{Verify}(vk, \sigma, m_k, t_k) = 1\end{array}\right] = 1 \tag{3}$$

Remark. Property 2 assumes that signatures and "forged" signatures used as input to the AltSign algorithm are computed honestly. One can also consider a stronger notion of correctness, where correctness of AltSign holds even on input signatures (and "forged" signatures) that may not be honestly computed, but nevertheless can be validated by the Verify algorithm. We refer to this as *robust correctness*.

In this work, we focus on the simpler notion and leave the discovery of schemes that satisfy robust correctness to future work.

Definition 5.3 (Robust Correctness)**.** A time-deniable signature scheme satisfies *robust correctness* if $\forall (vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T)$, $\forall (m_0, t_0, \sigma_0)$ such that $\mathsf{Verify}(vk, \sigma_0, m_0, t_0) = 1$, $\forall k \in \mathbb{N}$, for all sets of ordered tuples $\{(m_j, t_j)\}_{j\in[k]}$ where $\forall j \in [k]$, $t_{j-1} \geq t_j$:

$$\Pr\left[\begin{array}{l}\sigma \leftarrow \mathsf{AltSign}^k(vk, (m_0, t_0, \sigma_0),\\ \{(m_j, t_j)\}_{j\in[k]}) :\\ \mathsf{Verify}(vk, \sigma, m_k, t_k) = 1\end{array}\right] = 1 \tag{4}$$

5.1 Security Property: (ϵ, T)-Unforgeability

Our unforgeability notion requires that signatures should remain unforgeable within a restricted time window. We capture this via a security game below:

Setup. The challenger generates $(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T)$ and gives the verification key vk to the adversary.

Phase 1. The adversary is a tuple of two algorithms, $\mathcal{A}_0$ and $\mathcal{A}_1$. In this phase, $\mathcal{A}_0$ is allowed to adaptively query a signing oracle $\mathsf{O}_{\mathsf{Sign}}$ which is defined as follows. On input a message m and a timestamp t, the signing oracle $\mathsf{O}_{\mathsf{Sign}}(sk, \cdot, \cdot)$ returns the signature $\sigma \leftarrow \mathsf{Sign}(sk, m, t)$.

Transfer. The adversary $\mathcal{A}_0$ gives an advice string z to adversary $\mathcal{A}_1$.

Phase 2. The adversary $\mathcal{A}_1$ has to respond to the challenger with a forgery while also being allowed to adaptively query the oracle $\mathsf{O}_{\mathsf{Sign}}$.

Forgery. The adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ wins if in the end $\mathcal{A}_1$ can produce a valid forgery (m^*, t^*, σ^*) under the following constraints:

1. $\forall i \geq 1$, $t_i \geq t_{i-1}$ where (m_i, t_i) are the queries made by to $\mathsf{O}_{\mathsf{Sign}}$ by both $\mathcal{A}_0$ and $\mathcal{A}_1$
2. $\forall i, t^* > t_i$ for queries (m_i, t_i) made by $\mathcal{A}_0$
3. $\forall i, (m_i, t_i) \neq (m^*, t^*)$ where (m_i, t_i) are queries made to $\mathsf{O}_{\mathsf{Sign}}$ by $\mathcal{A}_1$

An adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is considered an ϵ-*admissible* adversary if it satisfies the above conditions and $\exists T(\lambda)$ where $\forall \lambda \in \mathbb{N}, \epsilon(\lambda) \leq T(\lambda)$, $\text{depth}(\mathcal{A}_1) \leq \frac{T(\lambda)}{\epsilon(\lambda)}$, and $\text{size}(\mathcal{A}) \in \text{poly}(\lambda)$. Note that the depth of $\mathcal{A}_0$ is allowed to be polynomial in the security parameter whereas the depth of $\mathcal{A}_1$ is more strictly bounded.

Definition 5.4 ((ϵ, T)-Unforgeability)**.** A time-deniable signature scheme satisfies the ϵ-Unforgeability property if $\forall$ ϵ-*admissible* polysize adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}} = \{(\mathcal{A}_{\lambda,0}, \mathcal{A}_{\lambda,1})\}_{\lambda\in\mathbb{N}}$, $\forall T(\lambda)$ satisfying the ϵ-admissability requirement for $\mathcal{A}$, there exist a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T(\lambda)),\\ (z) \leftarrow {\mathcal{A}_{\lambda,0}}^{\mathsf{O}_{\mathsf{Sign}}(sk,\cdot,\cdot)}(vk),\\ (m^*, t^*, \sigma^*) \leftarrow {\mathcal{A}_{\lambda,1}}^{\mathsf{O}_{\mathsf{Sign}}(sk,\cdot,\cdot)}(z) :\\ \mathsf{Verify}(vk, m^*, t^*, \sigma^*) = 1\end{array}\right] \leq \mu(\lambda) \tag{5}$$

Remark. The prior definition is more akin to EUF-CMA unforgeability than strong unforgeability since we disallow adversaries from submitting forgeries on message time-stamp pairs they have received in the past. We leave the question of achieving strong unforgeability to future work.

5.2 Deniability

Deniability in our scheme comes from the fact that after T sequential time-steps, anyone can forge a valid signature under the verification-key of the original signer. Consequently, a time-deniable signature scheme should ensure the indistinguishability of signatures generated via the Sign and the $\mathsf{AltSign}$ algorithms. Otherwise the original signer could not deny that it signed a message at a particular time. We present below a security game to capture this idea. Our notion would be meaningful even if the adversary did not have access to the signing key, but we give them it as well in order to capture more powerful attacker models.

We now define the security game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$:

Setup. The challenger generates $(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T)$ and gives both the verification key vk and the signing key sk to the adversary $\mathcal{A}$. They also initialize an empty table $\mathcal{T}$ and sample $\beta \xleftarrow{\$} \{0, 1\}$.

Query Phase. In this phase, the adversary is allowed to adaptively query a signing oracle $\mathsf{O}_{\mathsf{Sign}}(sk, \cdot, \cdot)$ and a challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot, \cdot, \cdot)$.

- $\mathsf{O}_{\mathsf{Sign}}(sk, \cdot, \cdot)$ takes as input a message m and a timestamp t to produce $\sigma \leftarrow \mathsf{Sign}(sk, m, t)$. It randomly chooses a new identifier id (not equal to any previously defined identifiers), records $(\mathsf{id}, m, t, \sigma)$ in $\mathcal{T}$, and returns (id, σ)
- $\mathsf{O}_{\mathrm{Ch}}(\cdot, \cdot, \cdot)$ takes as input a tuple of identifier, challenge message, and time-stamp (id, m, t). It checks $\mathcal{T}$ for id. If it is not present the output is $\perp$. Let m', t', σ' be the values associated with id. If $t > t'$ the output is also $\perp$. Compute $\sigma^0 \leftarrow \mathsf{Sign}(sk, m, t)$ and $\sigma^1 \leftarrow \mathsf{AltSign}(vk, (m', t', \sigma'), m, t)$. Finally it responds with σ^β.

Guess. The adversary outputs its guess β' for β. The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = \Pr[\beta' = \beta] - \frac{1}{2}$. The adversary $\mathcal{A}$ wins if $\beta' = \beta$.

Definition 5.5 (Deniability)**.** A signature scheme is considered to possess the deniability property if $\forall$ poly size adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ their advantage $\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ in the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$ game is negligible.

k-hop Deniability: We refer to the definition defined above as 1-hop deniability. It is reasonable to ask if indistinguishability still holds when comparing the output of Sign with applying the $\mathsf{AltSign}$ algorithm k times instead of just once. Intuitively, this notion could be stronger and offer more deniability via a larger pool of indistingishable forgeries.

A formal definition of the k-hop indistinguishability game $\mathbf{Exp}^{\mathsf{khop}}_{\mathsf{DS}}$ is given below:

Setup. This is the same as the (1-hop) key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$.

Query Phase. The adversary $\mathcal{A}$ has access to two oracles $\mathsf{O}_{\mathsf{Sign}}(\mathsf{sk}, \cdot, \cdot)$ and $\mathsf{O}_{\mathrm{Ch}}(\cdot, \cdot, \cdot, \cdot)$. $\mathsf{O}_{\mathsf{Sign}}$ is the same oracle as given in $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$.

$\mathsf{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot,\cdot)$ takes as input a challenge identifier id, one ordered set of messages and time-stamp tuples $\{(m_i,t_i)\}_{i\in[k-1]}$, and a message, time-stamp pair m^*,t^*. It checks that there exists a row in $\mathcal{T}$ with $(\mathsf{id},\cdot,\cdot,\cdot)$. Let m_0,t_0,σ_0 be the values associated with that row. It ensures that $\forall i\in[k-1], t_{i-1}\geq t_i$, and $t_{k-1}\geq t^*$. If any of these does not hold, the output is $\perp$. Compute:

$$\sigma^0 \leftarrow \mathsf{Sign}(sk, m^*, t^*)$$
$$\sigma^1 \leftarrow \mathsf{AltSign}^k(vk, (m_0,t_0,\sigma_0), \{(m_1,t_1),\ldots,(m_{k-1},t_{k-1}),(m^*,t^*)\})$$

σ^β is returned as the output.

Guess. This is again the same as the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$ game.

Definition 5.6 (k-hop Deniability)**.** A signature scheme achieves k-hop deniability property if $\forall$ poly size adversaries $\mathcal{A}=\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ their advantage in the $\mathbf{Exp}^{\mathsf{khop}}_{\mathsf{DS}}$ game is negligible.

Theorem 5.1. *Any time-deniable signature scheme satisfying the deniability property as defined in definition 5.5, also satisfies the k-hop deniability property as defined in definition 5.6.*

For a proof of Theorem 5.1 see Appendix C.1.

6 Delegatable Functional Signatures

In this section we define and construct delegatable functional signatures and define an additional key indistinguishability property for this primitive.

Functional Signatures. We start by recalling the notion of Functional Signatures as defined by Boyle, Goldwasser, and Ivan[9].

Definition 6.1. A functional signature scheme $\mathsf{FS}=(\mathsf{Setup},\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify})$ is a tuple of potentially probabilistic, polynomial time algorithms of the following form:

$\mathsf{Setup}(1^\lambda)\rightarrow mvk, msk$: On input the security parameter, this algorithm returns the master verification key mvk and the master signing key msk.

$\mathsf{KeyGen}(msk, f)\rightarrow sk_f$: On input the master signing key msk and a function f, this algorithm outputs a function-specific signing key sk_f.

$\mathsf{Sign}(sk_f, f, m)\rightarrow (f(m),\sigma)$: On input a function-specific signing key, a function f and a message m, this algorithm outputs $f(m)$ and a signature σ.

$\mathsf{Verify}(mvk, f(m), \sigma)\rightarrow\{0,1\}$: On input a master verification key mvk, a function $f()$ evaluated on message m and a signature, it outputs a bit.

Correctness. Correctness requires that any signature output from the Sign algorithm on a valid functional key and a message verifies correctly.More formally, for all supported functions f, for all messages m,

$$\Pr\left[\begin{array}{c} mvk, msk \leftarrow \mathsf{Setup}(1^\lambda);\\ sk_f \leftarrow \mathsf{KeyGen}(msk, f);\\ m^*,\sigma \leftarrow \mathsf{Sign}(sk_f, f, m):\\ \mathsf{Verify}(mvk, m^*, \sigma)=1 \end{array}\right] = 1$$

Security. For completeness, the unforgeability security game $\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{FS}}$ between a challenger and adversary $\mathcal{A}$ for functional signatures is provided below.

Setup. The challenger generates $(mvk, msk) \leftarrow \mathsf{Setup}(1^\lambda)$. They also initialize an empty table $\mathcal{T}$. mvk is given to adversary $\mathcal{A}$.

Query Phase. In this phase $\mathcal{A}$ gets access to a key oracle $\bar{\mathsf{O}}_{\mathsf{Key}}$ and a signing oracle $\bar{\mathsf{O}}_{\mathsf{Sign}}$.

1. $\bar{\mathsf{O}}_{\mathsf{Key}}(msk, \cdot, \cdot)$ takes as input function description f and an identifier i. The challenger checks if there is a row in $\mathcal{T}$ corresponding to $(i, f, \cdot)$. If such a row exists then return the corresponding secret key sk_f^i. Otherwise generate $sk_f \leftarrow \mathsf{KeyGen}(msk, f)$, record (i, f, sk_f) in $\mathcal{T}$ and return sk_f.
2. $\bar{\mathsf{O}}_{\mathsf{Sign}}(msk, \cdot, \cdot, \cdot)$ takes as input a function description f, an identifier i, and a message m. If a row in $\mathcal{T}$ corresponds to $(i, f, \cdot)$ then use the secret key sk_f specified in that row. Otherwise, generate $sk_f \leftarrow \mathsf{KeyGen}(msk, f)$ and record (i, f, sk_f) in $\mathcal{T}$. Let $f(m), \sigma$ be the output of $\mathsf{Sign}(sk_f, f, m)$. Return σ to $\mathcal{A}$.

Challenge Phase. The adversary $\mathcal{A}$ must return m^*, σ^* to the challenger. An adversary is considered to be *admissable* if that the following conditions are satisfied:

1. $\forall$ values (m_i, σ_i) returned by $\bar{\mathsf{O}}_{\mathsf{Sign}}$, $m_i \neq m^*$
2. there is no key sk_f queried from $\bar{\mathsf{O}}_{\mathsf{Key}}$ such that $\exists m$ where $f(m) = m^*$

A functional signature scheme is said to be secure if for all *admissable* poly-size adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ there exists a negligible function $\mu(\lambda)$ such that

$$\Pr\left[\begin{array}{l} mvk, msk \leftarrow \mathsf{KeyGen}(1^\lambda); \\ m^*, \sigma^* \leftarrow \mathcal{A}_\lambda^{\bar{\mathsf{O}}_{\mathsf{Key}}(msk,\cdot,\cdot), \bar{\mathsf{O}}_{\mathsf{Sign}}(msk,\cdot,\cdot,\cdot)}(mvk) \end{array} : \mathsf{Verify}(mvk, m^*, \sigma^*) = 1\right] \leq \mu(\lambda)$$

6.1 Key Delegation

In order to create signing keys even without the master signing key, we define an additional probabilistic polynomial time algorithm called $\mathsf{Delegate}$. This algorithm takes as input a function f, a corresponding secret key sk_f, and a *restriction* of f, f'. The output is a secret key $sk_{f'}$ or $\perp$. We say that a function f' is a *restriction* of another function f if the following is true: let $f : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$, then f' has the same domain and codomain as f and $\forall x \in \mathcal{X}$ either $f'(x) = f(x)$ or $f'(x) = \perp$. This captures the ability to create a signing key that can sign some subset of the same messages as the original key. Notice that restriction here is almost a misnomer since technically f can be a restriction of itself.

$\mathsf{FS.Delegate}(mvk, f, sk_f, f') \to sk_{f'}, \perp$: given the verification key mvk, a function f, a signing key sk_f, and another function f' output $sk_{f'}$ if f' is a restriction of f else $\perp$.

For a delegatable functional signature scheme the following additional correctness property must hold for all functions $f : \mathcal{X} \to \mathcal{Y} \cup \{\perp\}$ supported by FS, for all restrictions f' of f, and $\forall m \in \mathcal{X}$ where $f'(m) \neq \perp$:

$$\Pr\left[\begin{array}{l} (mvk, msk) \leftarrow \mathsf{FS.Setup}(1^\lambda); \\ sk_f \leftarrow \mathsf{FS.KeyGen}(msk, f); \\ sk_{f'} \leftarrow \mathsf{FS.Delegate}(mvk, f, sk_f, f'); \\ \sigma \leftarrow \mathsf{FS.Sign}(sk_{f'}, f', m) : \\ \mathsf{FS.Verify}(mvk, f'(m), \sigma) = 1 \end{array}\right] = 1 \tag{6}$$

The relevance of the delegation property will be demonstrated in our construction. Furthermore, our construction will require yet another property of these delegatable functional signatures.

Key Indistinguishability We would like it to be the case that keys generated via KeyGen and $\mathsf{Delegate}$ appear the same to any adversary, even if they have access to the master signing key msk and can make adaptive queries. To capture this notion we define the key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$ for delegatable functional signatures.

Setup The challenger runs $(mvk, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives both the master verification key mvk and the master signing key msk to the adversary. Let $\mathcal{T}$ be a table kept by the challenger, initialized to be empty. The challenger also samples $\beta \xleftarrow{\$} \{0,1\}$ and keeps this value to itself.

Query Phase In this phase, the adversary gets to query two different oracles.

1. Key creation oracle $\mathsf{O}_{\mathsf{Key}}(\cdot)$, which can be queried on some specific function f. On input a function f, the key creation oracle checks $\mathcal{T}$ for keys on function f. Let i be the largest value associated with a row containing f. Run $sk_f \leftarrow \mathsf{FS.KeyGen}(msk, f)$ and record $(i+1, f, sk_f)$ in $\mathcal{T}$. Output $(i+1, sk_f)$.
2. Challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ where the first input is an identifier i and the subsequent inputs are functions f_0 ,f_1 and f_1 is a restriction of f_0. The challenger checks $\mathcal{T}$ for a row $(i, f_0, \cdot)$ that has secret key sk_{f_0}. If no such key exists, the output is $\perp$. Otherwise, the oracle computes $sk_0 = \mathsf{FS.Delegate}(mvk, f_0, sk_{f_0}, f_1)$, $sk_1 = \mathsf{FS.KeyGen}(msk, f_1)$ and returns sk_β.

Guess The adversary outputs its guess β' for β and wins if $\beta' = \beta$.

The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = \left|\Pr\left[[\beta' = \beta] - \frac{1}{2}\right]\right|$.

Definition 6.2 (Key-Indistinguishability for Delegatable FS)**.** A delegatable functional signatures scheme is considered key-indistinguishable if $\forall$ poly size adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ their advantage $\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ in the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$ game is negligible.

6.2 Construction for Prefix Functions

We now describe how to create delegatable functional signatures for prefixing functions from hierarchical identity-based encryption. We will be concerned with signatures on functions of the form $f_T : \{0,1\}^\ell \times \{0,1\}^m \to \{0,1\}^{\ell+m}$ that concatenate their arguments. More formally, we consider functions

$$f_T(t, m) = \begin{cases} t||m & t \le T \\ \perp & \text{otherwise} \end{cases} \tag{7}$$

For the sake of readability, in the following constructions we abuse notation and write T in place of f_T i.e. $\mathsf{FS.Delegate}(mvk, f_y, sk_{f_y}, f_{y'})$ is replaced with $\mathsf{FS.Delegate}(mvk, y, sk_y, y')$.

We also define the notion of *stack trace* which will be useful in the formal description of the protocol.

Definition 6.3. The stack trace of T, $\mathsf{Trace}(r, T)$ is defined as the set of nodes on the stack when executing a depth-first search to find the leaf node $T+1$ in a binary tree with some root r.

The stack trace can be found efficiently, and as described in the technical overview gives us the set of the $\le \ell$ identity key nodes required to derive all keys corresponding to timestamps up to T.

The construction is presented in pseudocode in Figure 2.

Theorem 6.1. *If* HIBE *is adaptively secure then the functional signature scheme for prefix functions constructed in Figure 2 is unforgeable.*

For a proof of Theorem 6.1 see Appendix B.

Theorem 6.2. *If* HIBE *is key-indistinguishable then the scheme in Figure 2 satisfies the functional signatures key-indistinguishability property.*

For a proof of Theorem 6.2 see Appendix B.2.

$\underline{\mathsf{Setup}(1^\lambda):}$
$(msk', pk) \leftarrow \mathsf{HIBE.Setup}(1^\lambda)$
return $pk, (pk, msk')$

$\underline{\mathsf{KeyGen}(msk = (pk, msk'), T):}$
$\text{list}_{sk_T} := []$
$trace \leftarrow \mathsf{Trace}(msk, T)$
for $node \in trace$:
  $sk_{node} \leftarrow \mathsf{HIBE.KeyGen}(msk', node)$
  $\text{add}(\text{list}_{sk_T}, sk_{node})$
return pk, list_{sk_T}

$\underline{\mathsf{Sign}(sk_T, T, (t, m)):}$
if $t > T$:
  return $\perp$
$t_1 \dots t_\ell \leftarrow \text{parse}(t)$
$pk, list_{sk_T} \leftarrow \text{parse}(sk_T)$
$sk_{t'}, j \leftarrow \mathsf{findPrefix}(list_{sk_T}, t)$
$sk_{t||m} \leftarrow \mathsf{HIBE.Delegate}(pk, sk_{t'}, t_{j+1} \dots t_\ell || m)$
return $t||m, sk_{t||m}$

$\underline{\mathsf{Delegate}(mvk = pk, T, sk_T = (pk', \text{list}_{sk_T}), T'):}$
if $T' > T$:
  return $\perp$
$T'_1 \dots T'_\ell \leftarrow \text{parse}(T')$
$sk, j \leftarrow \mathsf{findPrefix}(\text{list}_{sk_T}, T')$
$\text{list}_{sk_{T'}} := []$
for $i \in 0, \dots, j-1$:
  $sk_i \leftarrow \mathsf{HIBE.Delegate}(pk, \text{list}_{sk_T}[i], nil)$
  $\text{add}(\text{list}_{sk'_T}, sk_i)$
$trace \leftarrow \mathsf{Trace}(T'_1 \dots T'_j, T'_l)$
for $node \in trace$:
  $sk_{node} \leftarrow \mathsf{HIBE.Delegate}(pk, sk, node)$
  $\text{add}(\text{list}_{sk_{T'}}, sk_{node})$
return $pk, \text{list}_{sk_{T'}}$

$\underline{\mathsf{Verify}(mvk = pk, m^*, \sigma = sk_{t||m}):}$
$msg \leftarrow \{0,1\}^n$
$c \leftarrow \mathsf{HIBE.Encrypt}(pk, msg, sk_{t||m})$
return $\mathsf{HIBE.Decrypt}(sk_{t||m}, c) = msg$ and $t||m = m^*$

Figure 2: The function f_T for each input message m is defined as $f_T(t, m) = \mathsf{parse}(t) \| m$ if $t \leq T$ or $\perp$.$\mathsf{findPrefix}(list, id)$ takes in a list of HIBE secret keys called $list$ and an identity string id. It returns a secret key sk and an index j so that sk is a secret key for a length j prefix of id. Any bit string beginning with t_{j+1} where $j = \ell$ is the empty string. The function $\mathsf{Trace}(root, leaf)$ is specified in definition 6.3.

7 Construction of Time-Deniable Signatures

This section describes our construction of time-deniable signatures from key indistinguishable, delegatable functional signatures for prefix functions and time lock encryption. To sign a message at time stamp t, we first use the master signing key to construct a signing key for the function f_t. This key is then used to sign the message and is included as a ciphertext, time-lock encrypted with the signature. The alternate signing algorithm decrypts the ciphertext, uses the delegate algorithm to produce an appropriate signing key for $f_{t'}$ with $t' \leq t$, and then signs and time-lock encrypts the key. For security of the scheme to hold, the parameter for the time-lock puzzle Δ cannot be precisely the same as T. The intuitive reason behind the difference is that forging involves not just breaking the time lock but also executing other algorithms. Let $|\mathsf{A.B}|$ denote the depth of the circuit that computes algorithm B of cryptographic primitive A and $z(\lambda) = |\mathsf{FS.Verify}| + 2 + 1 + |\mathsf{FS.Sign}| + |\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}|$. Our construction is described in Figure 3 and uses $z(\lambda)$ to define Δ.

Theorem 7.1. *Consider the scheme presented in figure 3. Let $\mathcal{A} = \{(\mathcal{A}_{\lambda,0}, \mathcal{A}_{\lambda,1})\}_{\lambda \in \mathbb{N}}$ be a polysize adversary in the (ϵ, T)-unforgeability game, $q(\lambda)$ the max number of queries $\mathcal{A}_{\lambda,1}$ can make, m the number of queries $\mathcal{A}_{\lambda,1}$ actually makes, $\mathcal{B}$ an adversary against the ϵ-gap time lock puzzle scheme, and $\mathcal{C}$ an adversary against the adaptive unforgeability of the functional signature scheme. Then the following holds:*

$$\mathbf{Adv}_A^{\epsilon-\mathsf{UF}}(\lambda) \leq (q(\lambda)+1) \cdot \left(m \cdot \mathbf{Adv}_{\mathcal{B}}^{\epsilon-\mathsf{TimeLock}}(\lambda) + \mathbf{Adv}_{\mathcal{C}}^{\mathsf{FS}}(\lambda)\right)$$

For a proof of Theorem 7.1 see Appendix A.

Theorem 7.2. *If the underlying delegatable functional signature scheme is key-indistinguishable then the constructed time-deniable signatures scheme satisfies the deniability property.*

$\underline{\mathsf{KeyGen}(1^\lambda, T):}$	$\underline{\mathsf{AltSign}(vk, (m^*, t^*, \sigma^*), m, t):}$	$\underline{\mathsf{Verify}(vk, \sigma, m, t):}$
$(mvk, msk) \leftarrow \mathsf{FS.Setup}(1^\lambda)$	$mvk, T, \lambda = parse(vk)$	$c, s = \mathsf{parse}(\sigma)$
return $((mvk, T, \lambda), (msk, T, \lambda))$	$c^*, s^* = \mathsf{parse}(\sigma^*)$	return $\mathsf{FS.Verify}(vk, t\|\|m, s)$
	$sk_{t^*} = \mathsf{TimeLock.Sol}(c^*)$	
$\underline{\mathsf{Sign}(sk = (msk, T, \lambda), m, t):}$	$sk_t \leftarrow \mathsf{FS.Delegate}(mvk, f_t^*, sk_t^*, f_t)$	
$sk_t \leftarrow \mathsf{FS.KeyGen}(msk, f_t)$	$v, s \leftarrow \mathsf{FS.Sign}(sk_t, f_t, (t, m))$	
$v, s \leftarrow \mathsf{FS.Sign}(sk_t, f_t, (t, m))$	$c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T \cdot z(\lambda), sk_t)$	
$c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T \cdot z(\lambda), sk_t)$	return (c, s)	
return (c, s)		

Figure 3: A construction for a time-deniable signature scheme DS from a key-indistinguishable, delegatable, functional signature scheme FS and a time lock puzzle TimeLock. The function f_t for each input message m and time $\bar{t}$ is defined as $f_t(\bar{t}, m) = \mathsf{parse}(\bar{t})\|m$ if $\bar{t} \leq t$, else $\perp$. The polynomial $z(\lambda)$ is a multiplicative factor for the difficulty parameter of the time lock puzzle and is described in the text.

For a proof of Theorem 7.2 see Appendix A.

8 Implementation and Evaluation

Implementation. To demonstrate the feasibility of our scheme we implemented it in python by modifying existing implementations of a RSW time-lock puzzle for TimeLock and a scheme of Lewko-Waters's for HIBE [22, 28]. For our supported time-stamp lengths we use $\ell = 32$. This is reasonable as the granularity of a UNIX timestamp is also 32 bits. We modified the HIBE to support key re-randomization and delegation from level i to level $i + k$ for $k \geq 1$. Small optimizations we implemented include using the hash and sign paradigm, multi-threading for independent group operations, and using an n-ary tree for HIBE identities, the latter of which introduces a trade-off between signature size and computation time.

Experimental Evaluation. Experiments were done on a laptop with a 2GHz Quad-Core Intel Core i5 processor and 32 GB of RAM running macOS Big Sur. When instantiating it is necessary to translate sequential computation time steps to wall clock time. We calculated the time of sequential squaring using our test laptop to be roughly 5,883,206 per second. In practice, this bound would be calculated considering the best available hardware. Figure 4 shows the computation time of the Sign algorithm for example small unforgeability periods. As expected by construction, the time to sign a message does not grow with the unforgeability window which is highly desirable. Future implementations of this protocol could utilize a more efficient HIBE scheme such as [17] and do other optimizations to achieve better performance.

9 On the Necessity of Time-Lock Puzzles

Our construction of time-deniable signatures makes uses of time-lock puzzles to achieve short-term unforgeability. We show that the use of such a primitive is to an extent inherent. Namely, assuming extractable witness encryption [15, 18], we show that time-deniable signatures imply time-lock puzzles.

We demonstrate this implication in Appendix E. We remark that while extractable witness encryption is a strong tool, it alone is not known to imply time-lock puzzles.[8]

References

[1] Signal protocol documentation. `https://www.signal.org/docs/`.

[8] When supplemented with a computational reference clock, it is known to imply time lock puzzles [26].

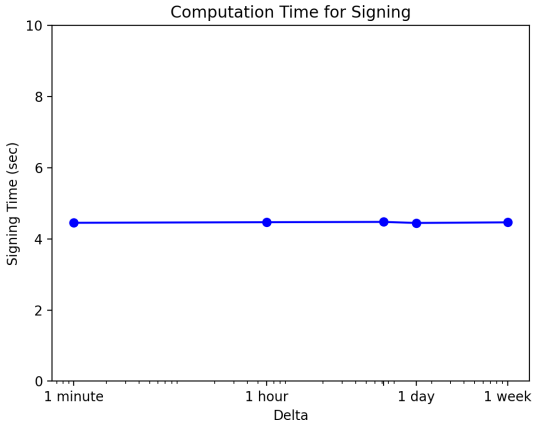

Figure 4: This figure shows the cost of running the Sign algorithm for different supported unforgeability periods T. The scale of the x axis is logarithmic. Each number was obtained by 100 signing attempts on the same message with the unix time-stamp 1634098632. The identity tree was 17-ary of depth 8.

[2] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures.

[3] Michael Backes, Sebastian Meiser, and Dominique Schröder. Delegatable functional signatures. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part I*, volume 9614 of *LNCS*, pages 357–386. Springer, Heidelberg, March 2016.

[4] Mihir Bellare and Georg Fuchsbauer. Policy-based signatures. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 520–537. Springer, Heidelberg, March 2014.

[5] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

[6] 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.

[7] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84, 2004.

[8] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

[9] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*, pages 501–519. Springer, 2014.

[10] Jon Callas, Eric Allman, Miles Libbey, Michael Thomas, Mark Delany, and Jim Fenton. Domainkeys identified mail (dkim) signatures.

[11] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. Bounded CCA2-secure encryption. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 502–518. Springer, Heidelberg, December 2007.

[12] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Non-malleable time-lock puzzles and applications. *IACR Cryptol. ePrint Arch.*, 2020:779, 2020.

[13] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. SPARKs: Succinct parallelizable arguments of knowledge. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 707–737. Springer, Heidelberg, May 2020.

[14] Ethereum Foundation . The Ethereum Beacon Chain. Available at `https://ethereum.org/en/eth2/beacon-chain/`.

[15] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[16] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[17] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, pages 548–566, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[18] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013.

[19] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012.

[20] Matthew D. Green. Ok google: Please publish your dkim secret keys, Dec 2020.

[21] Andreas Hülsing and Florian Weber. Epochal signatures for deniable group chats. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1677–1695, 2021.

[22] Jonathan Levi. time-lock-puzzle. Available at `https://github.com/drummerjolev/time-lock-puzzle`, 2020.

[23] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 318–335. Springer, Heidelberg, April 2012.

[24] Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 547–567, 2011.

[25] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 58–76. Springer, Heidelberg, May 2014.

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

[27] Jeremy B. Merrill. Authenticating email using dkim and arc, or how we analyzed the kasowitz emails, Jul 2017.

[28] Nikos Fotiou. Hibe-lw11. Available at `https://github.com/nikosft/HIBE_LW11`, 2014.

[29] NIST. NIST Randomness Beacon (prototype implementation). Available at `https://beacon.nist.gov/home`.

[30] Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 59:1–59:25. LIPIcs, January 2019.

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

[32] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[33] Raphael Satter. Emails: Lawyer who met trump jr. tied to russian officials, Jul 2018.

[34] Michael A. Specter, Sunoo Park, and Matthew Green. Keyforge: Non-attributable email from forward-forgeable signatures. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1755–1773, 2021.

[35] Nik Unger and Ian Goldberg. Improved strongly deniable authenticated key exchanges for secure messaging. *Proc. Priv. Enhancing Technol.*, 2018(1):21–66, 2018.

[36] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

A Proofs for Time Deniable Signatures

Theorem A.1. *The time-deniable signature scheme presented in figure 3 is unforgeable.*

In the discussion that follows, let the output of a hybrid game $\mathcal{H}$ be the output of the challenger. We prove the theorem statement using a hybrid argument where $\mathcal{H}_0$ represents the original (ϵ, T)-unforgeability game. Where details are omitted in the hybrid description of $\mathcal{H}_i$, it is assumed they are the same as in $\mathcal{H}_{i-1}$.

$\mathcal{H}_1$ = Let $q(\lambda) = \mathsf{size}(\mathcal{A}_{\lambda,1})$. Challenger samples $r \xleftarrow{\$} [q(\lambda)] \cup \{0\}$. If the number of queries made to $\mathsf{O_{Sign}}$ by $\mathcal{A}_{\lambda,1}$ is not r, output a random bit b'.

Claim 1. $\mathbf{Adv}^{\mathcal{H}_1}_{\mathcal{A}}(\lambda) = \frac{1}{q(\lambda)+1}\mathbf{Adv}^{\mathcal{H}_0}_{\mathcal{A}}(\lambda)$

Note that the win condition is checked whenever the challenger"correctly guesses" how many queries will be made by the adversary in the second phase. Let m be the number of queries made by $\mathcal{A}_{\lambda,1}$, where $0 \leq m \leq q(\lambda)$. This must hold since the adversary cannot make more queries than its size dictates. Therefore,

$$\mathbf{Adv}^{\mathcal{H}_1}_{\mathcal{A}}(\lambda) = \Pr(r = m | r \xleftarrow{\$} \{0, \ldots, q(\lambda)\})\mathbf{Adv}^{\mathcal{H}_0}_{\mathcal{A}}(\lambda) \tag{8}$$

It should be clear the first quantity is $\frac{1}{q(\lambda)+1}$ and the claim is thus true.

Consider the following sequence of hybrids where $2 \leq i \leq m+1$

$\mathcal{H}_i$ = On the $(i-1)^{th}$ query to $\mathsf{O_{Sign}}$ by $\mathcal{A}_{\lambda,1}$, challenger replaces $c = \mathsf{TimeLock.Gen}(T, sk_t)$ with $\mathsf{TimeLock.Gen}(T, 0)$.

Claim 2. $\exists \mathcal{B}$, *a* $\epsilon - \mathsf{TimeLock}$ *adversary, such that* $|\mathbf{Adv}^{\mathcal{H}_i}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{\mathcal{H}_{i-1}}_{\mathcal{A}}(\lambda)| \leq \mathbf{Adv}^{\epsilon-\mathsf{TimeLock}}_{\mathcal{B}}(\lambda)$

WLOG, assume $\mathbf{Adv}^{\mathcal{H}_i}_{\mathcal{A}} > \mathbf{Adv}^{\mathcal{H}_{i-1}}_{\mathcal{A}}$. First consider the following distinguisher $\mathcal{D}$ between $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$, where output of $\mathcal{D}$ being 0 denotes $\mathcal{H}_{i-1}$ and 1 denotes $\mathcal{H}_i$.

Description of $\mathcal{D}$ on input b from $\mathcal{H}_{i-1}$ or $\mathcal{H}_i$:

- If $b = 1$ output $b' = 1$
- If $b = 0$, output $b' = 0$.

Notice that $\mathcal{D}$'s advantage in distinguishing is dependent on $\mathbf{Adv}^{\mathcal{H}_i}_{\mathcal{A}}(\lambda) - \mathbf{Adv}^{\mathcal{H}_{i-1}}_{\mathcal{A}}(\lambda)$ and that the depth of $\mathcal{D}$ is 2. We will now use $\mathcal{D}$ and $\mathcal{A}_\lambda$ to construct an adversary $\mathcal{B}'$.

Description of $\epsilon - \mathsf{TimeLock}$ adversary $\mathcal{B}'$:

- Honestly run the $\mathsf{FS.Setup}$ algorithm and answer all queries from $\mathcal{A}_{\lambda,0}$ honestly
- For the j^{th} query from $\mathcal{A}_{\lambda,1}$:
  - if $j < i-1$, $c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T, 0)$
  - if $j > i-1$, $c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T, sk_{t_j})$
  - if $j = i-1$, compute $sk_{t_{i-1}} \leftarrow \mathsf{FS.KeyGen}(msk, f_{t_{i-1}})$, send the challenger $(sk_{t_{i-1}}, 0)$, and receive z. Set $c = z$.
- From $\mathcal{A}_\lambda$ get output b and give b to $\mathcal{D}$. Get b' from $\mathcal{D}$. Output b' to the challenger.

Define $(\mathcal{B}_1, \mathcal{B}_2) = \mathcal{B}'$ where $\mathcal{B}_1$ represents $\mathcal{B}'$ up until query $i-1$ is made and $\mathcal{B}_2$ is all that follows. Let the output of $\mathcal{B}_1$ be z and hard-code it into $\mathcal{B}_2$ to get $\mathcal{B}$.

Analysis of adversary $\mathcal{B}$:

Let $|\mathsf{A.B}|$ denote the depth bound on the algorithm B for primitive A. The depth bound for $\mathcal{B}$ is represented below

$$\begin{aligned}
\mathsf{depth}(\mathcal{B}) &\leq \mathsf{depth}(\mathcal{A}_1) + m \cdot \Big(|\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}| + |\mathsf{FS.Sign}|\Big) + |\mathsf{FS.Verify}| + 2 \\
&\leq \frac{t(\lambda)}{\epsilon(\lambda)} + m \cdot \Big(|\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}| + |\mathsf{FS.Sign}|\Big) + |\mathsf{FS.Verify}| + 2 \\
&\leq \frac{\epsilon(\lambda)(|\mathsf{FS.Verify}| + 2)}{\epsilon(\lambda)} + \frac{t(\lambda)[1 + |\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}| + |\mathsf{FS.Sign}|]}{\epsilon(\lambda)} \\
&\leq \frac{t(\lambda)z(\lambda)}{\epsilon(\lambda)}
\end{aligned}$$

Assuming the ϵ-gap security of the $\mathsf{TimeLock}$ and because $\Delta = t'(\lambda) = t(\lambda) \cdot z(\lambda)$ in our construction, $\mathcal{B}$ is appropriately bounded.

$$\begin{aligned}
\Pr(\mathcal{B} \text{ succeeds}) &= \frac{1}{2}\Pr(\mathcal{B} \text{ succeeds} \mid \beta = 0) + \frac{1}{2}\Pr(\mathcal{B} \text{ succeeds} \mid \beta = 1) \\
&= \frac{1}{2}\Pr(\mathcal{D} \text{ outputs } 0 \text{ when given } \mathcal{H}_{i-1}) + \frac{1}{2}\Pr(\mathcal{D} \text{ outputs } 1 \text{ when given } \mathcal{H}_i) \\
&= \Pr(\mathcal{D} \text{ correctly distinguishes } \mathcal{H}_{i-1} \text{ and } \mathcal{H}_i)
\end{aligned}$$

$\mathcal{B}$'s probability of success entirely depends on $\mathcal{D}$'s and thus $\mathcal{B}$'s advantage is the same as $\mathcal{D}$'s. The claim thus follows.

Claim 3. *$\exists \mathcal{C}$, an adversary against the unforgeability of the FS scheme, s.t. $\mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_{m+1}}(\lambda) \leq \mathbf{Adv}_{C}^{\mathsf{FS}}(\lambda)$*

We now argue that the advantage of any adversary in $\mathcal{H}_{m+1}$ can be translated into equivalent advantage against the unforgeability scheme.

Description of $\mathcal{C}$:

- Receive mvk from the FS challenger.
- On queries m, t to $\mathsf{O}_{\mathsf{Sign}}(\cdot)$
  - if phase one, query $\bar{\mathsf{O}}_{\mathsf{Key}}(f_t, i)$ and receive sk_{f_t} where $i \in \mathbb{N}$ is next available counter. Compute $\sigma \leftarrow \mathsf{FS.Sign}(sk_{f_t}, f_t, (t, m))$, $c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T, sk_{f_t})$. Return (c, σ) to $\mathcal{A}_{\lambda,0}$
  - if phase two, query $\bar{\mathsf{O}}_{\mathsf{Sign}}(f_t, i, (t, m))$ with $i \in \mathbb{N}$ being the next highest counter to get m^*, σ. Compute $c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, T, 0)$ and return (c, σ) to $\mathcal{A}_{\lambda,1}$
- If $\mathcal{A}_\lambda$ returns forgery $m, t, \sigma = (c, s)$ return $(t||m, s)$ to the FS challenger.

Analysis. We now show that if $\mathcal{A}$ is successful, then $\mathcal{C}$ must be as well. Say $\mathcal{A}$ returns a forgery $m^*, t^*, \sigma^* = (c, s)$. In order for $\mathcal{A}$ to be admissable, it must be true that $\mathcal{A}$ never received a signature with $t \geq t^*$ during the first phase and during the second phase there was never a query for (m^*, t^*) specifically. The first point implies $\mathcal{C}$ never queries for a secret key for a function f_t where $t \geq t^*$ so s is a valid signature to give back to the functional challenger. The second point means that $\mathcal{A}$ is not giving $\mathcal{C}$ a signature that $\mathcal{C}$ asked for from the FS challenger with some mauled c' where c' is an incorrectly structured puzzle or does not hide the right secret key. Therefore if $\mathcal{A}$ returns a valid forgery, then $\mathcal{C}$ returns a valid forgery and the claim follows.

Theorem A.2. *If the underlying delegatable functional signature scheme is key-indistinguishable then the constructed time-deniable signatures scheme satisfies the deniability property.*

We prove this by showing how to use an adversary $\mathcal{A}$ who wins the time-deniable signatures $\mathbf{Exp}_{\mathsf{DS}}^{\mathsf{IND}}$ game to construct an adversary $\mathcal{B}$ which wins the delegatable functional signatures key-indistinguishability game $\mathbf{Exp}_{\mathsf{FS}}^{\mathsf{IND}}$.

- $\mathcal{B}$ receives the master verification key and master signing key (mvk, msk) from its challenger and forwards it as is to $\mathcal{A}$.
- In the query phase, whenever $\mathcal{A}$ makes an $\mathsf{O}_{\mathsf{Sign}}(sk, \cdot, \cdot)$ query on message m and timestamp t, $\mathcal{B}$ performs the following operations:
  - Query the FS key oracle $\bar{\mathsf{O}}_{\mathsf{Key}}(msk, \cdot, \cdot)$ on t to get back (id, sk_t). Add the response to its internal table of responses. Otherwise, return $\perp$.
  - Use the received key to create $v, \sigma_{\mathsf{FS}} \leftarrow \mathsf{FS.Sign}(f_t, sk_t, t, m)$ and $c \leftarrow \mathsf{TimeLock.Gen}(\Delta, sk_t)$.
  - Let $\sigma_{\mathsf{DS}} = (c, \sigma_{\mathsf{FS}})$. Send $(\mathsf{id}, \sigma_{\mathsf{DS}})$ to $\mathcal{A}$. This simulates the functionality of $\mathsf{O}_{\mathsf{Sign}}(sk, t, m)$ which outputs an identifier and $\sigma \leftarrow \mathsf{DS.Sign}(sk, m, t)$.
- In the query phase, whenever $\mathcal{A}$ makes a query to the DS challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot, \cdot, \cdot)$ on some tuple (id', m', t'), $\mathcal{B}$ performs the following operations:
  - Query the FS challenge oracle on (id', t, t') if id' corresponds to a query on time t in its table and $t' < t$.
  - Using the received key $sk_{t'}$, compute $v', \sigma'_{\mathsf{FS}} \leftarrow \mathsf{FS.Sign}(f_{t'}, sk_{t'}, t, m)$ and $c' \leftarrow \mathsf{TimeLock.Gen}(\Delta, sk_{t'})$.
  - Send $\sigma'_{\mathsf{DS}} = (c', \sigma'_{\mathsf{FS}})$ to $\mathcal{A}$. This simulates the $\mathsf{DS.Sign}()$ algorithm when the challenger's sampled bit β is 0 as the key used for $\mathsf{FS.Sign}()$ is a freshly generated key. When $\beta = 1$, the key used for $\mathsf{FS.Sign}()$ is a delegated key, which simulates the $\mathsf{DS.AltSign}()$ algorithm.
- At the end, $\mathcal{A}$ outputs its guess β' for β, $\mathcal{B}$ forwards this without change to its challenger. The advantage of $\mathcal{B}$ in winning the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$ game is same as the advantage of $\mathcal{A}$ in winning the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$ game as all the responses to $\mathcal{A}$'s queries are simulated correctly by $\mathcal{B}$.

B Proofs for Delegatable Functional Signatures

Theorem B.1. *If* HIBE *is adaptively secure then the functional signature scheme for prefix functions constructed in Figure 2 is unforgeable.*

We prove this by showing how to use an adversary $\mathcal{A}$ who succeeds with non-negligible advantage in $\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{FS}}$ to construct an adversary $\mathcal{B}$ which succeeds with non-negligible advantage in the HIBE unforgeability game.

Description of $\mathcal{B}$

- In the setup phase, initialize empty table $\mathcal{T}$ and receive pk from HIBE challenger and forward it to $\mathcal{A}$.
- When $\mathcal{A}$ queries (f_t, i) to $\bar{\mathsf{O}}_{\mathsf{Key}}$ check if row with i in $\mathcal{T}$.
  - If it exists, return the list of keys sk_t to $\mathcal{A}$.
  - Otherwise, use the algorithm $\mathsf{FS.KeyGen}$ algorithm in Figure 2 replacing $\mathsf{HIBE.KeyGen}(msk, node_{\mathsf{id}})$ with $\mathcal{QC}(\mathsf{id})$ and $\mathcal{QR}(\mathsf{id})$. Let $list_{sk_t}$ be the resulting list of keys. Record $(i, t, list_{sk_t})$ in $\mathcal{T}$. Send pk, list_{sk} to $\mathcal{A}$.
- When $\mathcal{A}$ queries (f_t, i, m) to $\bar{\mathsf{O}}_{\mathsf{Sign}}$ first parse m as $\bar{t}||\bar{m}$. If $\bar{t} > t$ output $\perp$. Check if there is row in $\mathcal{T}$ with identifier i
  - If there is, let $list_{sk_t}$ be the list of keys associated with that row. Let $sk_{t'}$ be the key in $list_{sk_t}$ associated with an identity that is the prefix of $\bar{t}$. Compute $sk_{\bar{t}\|\bar{m}} \leftarrow \mathsf{HIBE.Delegate}(pk, sk_{t'}, \mathrm{suffix}(t', \bar{t}\|\bar{m})$ where suffix omits the prefix t' from $\bar{t}\|\bar{m}$. Return $sk_{\bar{t}\|\bar{m}}$

– Otherwise, use the algorithm $\mathsf{FS.KeyGen}$ in Figure 2 replacing $\mathsf{HIBE.KeyGen}(msk, node_{\mathsf{id}})$ with $\mathcal{QC}(\mathsf{id})$. Finally query $\mathcal{QD}(\bar{t}\|\bar{m})$ and do a subsequent reveal query $\mathcal{QR}(\bar{t}\|\bar{m})$ to get $sk_{\bar{t}\|\bar{m}}$. Return $sk_{\bar{t}\|\bar{m}}$ to $\mathcal{A}$.

- When $\mathcal{A}$ outputs its forgery (m^*, σ^*) parses m^* as $t\|m$ and σ^* as sk^*.
  - Sample a message msg and check that $\mathsf{Decrypt}(sk^*, \mathsf{HIBE.Encrypt}(pk, t\|m, msg) = msg$. If this does not hold or if $\exists$ a row $(i', t', list'_{sk_t})$ in $\mathcal{T}$ where $t \leq t'$ output $\beta' \xleftarrow{\$} \{0,1\}$
  - Otherwise randomly sample messages m_0 and m_1. Let $I^* = t\|m$. Send to challenger (m_0, m_1, I^*) and receive ct. Compute $m' \leftarrow \mathsf{HIBE.Decrypt}(sk_{I^*}, ct)$. If $m = m_0$, output 0. If $m = m_1$, output 1. If the response is neither, output $\beta' \xleftarrow{\$} \{0,1\}$.

Analysis

In order to be an admissable adversary, $\mathcal{A}$ must return a signature σ^* and an m^* that verify where they do not hold a functional key that has m^* in its range. The keys that have m^* in their range are of the form f_T where $T \geq t$. In other words, these functional keys are those that contain some HIBE secret keys that are prefixes of the identity t and no other functional key has such prefix HIBE keys by the design of the construction. Therefore if $\mathcal{A}$ is admissible, $m^* = t\|m$ will be a valid identity to challenge on.

If $\mathcal{A}$ is successful, then σ^* passed verification meaning for a random message it acted as a secret key for the identity $t\|m$. This implies with high confidence that it is in fact the secret key for this identity. Decrypting with the secret key for identity $t\|m$ the ciphertext ct will be successful with overwhelming probability and therefore most of the time when $\mathcal{A}$ succeeds $\mathcal{B}$ succeeds as well.

In any other circumstance, when $\mathcal{A}$ is either not admissible or does not return a valid forgery, $\mathcal{B}$ catches this and responds with a uniform bit. Thus the theorem statement follows.

Theorem B.2. *If* HIBE *is key-indistinguishable then the scheme in Figure 2 satisfies the functional signatures key-indistinguishability property.*

We prove this by showing how to use an adversary $\mathcal{A}$ who wins the delegatable functional signatures key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$ to construct an adversary $\mathcal{B}$ which wins the HIBE key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{HIBE}}$.

- $\mathcal{B}$ receives the keys (pk, msk) from its challenger and forwards it as is to $\mathcal{A}$.
- After this, in the query phase, when $\mathcal{A}$ makes a $\bar{\mathsf{O}}_{\mathsf{Key}}(\cdot)$ query for time t, adversary $\mathcal{B}$ computes $trace \leftarrow \mathsf{Trace}(root, t)$ where $root$ is the position of msk in the HIBE hierarchy tree. This gives $\mathcal{B}$ the list of nodes on which it queries the key oracle $\mathcal{QK}(\cdot)$. Each of the $\mathcal{QK}(\cdot)$ query response has an identifier id along with the key for a node sk_{node}. $\mathcal{B}$ maintains a table with entries of the form $(t, \mathsf{id}', \{(\mathsf{id}, sk_{node})\}_{node \in trace})$ where id' represents the counter value corresponding to this particular query from $\mathcal{A}$. $\mathcal{B}$ sends $(\mathsf{id}', sk_t = \{sk_{node}\}_{node \in trace})$ to $\mathcal{A}$. This is the response $\mathcal{A}$ expects as $\mathcal{B}$ simulates the $\mathsf{FS.KeyGen}(msk, t)$ algorithm with its queries to the HIBE key oracle.
- When $\mathcal{A}$ makes a FS challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ query with a tuple of the form (i, t_0, t_1), $\mathcal{B}$ performs the following operations:
  - Check that $t_0 > t_1$ and there is a row starting with (i, t_0) in its table, otherwise return $\perp$. This guarantees that on the shortest paths from the leaf node t_1 to the root in the HIBE hierarchy tree (Figure 1), there exists an element j such that its corresponding HIBE key is present in the set sk_{t_0} representing the FS key for t_0.
  - Compute $sk, j \leftarrow \mathsf{findPrefix}(sk_t, t_1)$ and $trace' \leftarrow \mathsf{Trace}(t_0^j, t_1)$ which is the trace of leaf node t_1 in a tree where the root is t_0^j, the first j bits of t_0. Rerandomize the key sk by computing $sk' \leftarrow \mathsf{HIBE.Delegate}(pk, sk, nil)$, similarly rerandomize all the keys in set sk_{t_0} upto the j'th position.

- Query the HIBE challenge oracle $\mathsf{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ on tuples $(\mathsf{id}_j, t_0^j, node)$ for all $node \in trace'$ where id_j corresponds to the $\mathcal{QK}(\cdot)$ response on t_0^j. $\mathcal{B}$ finds id_j in its table, in the row corresponding to t_0.
- Compile all the rerandomized keys and the keys received from the key oracle into set sk_{t_1}. Send sk_{t_1} to $\mathcal{A}$. This is the response $\mathcal{A}$ expect as $\mathcal{B}$ simulates either $\mathsf{FS.KeyGen}()$ or $\mathsf{FS.Delegate}()$ depending on the challenger's sampled bit β.

- At the end, $\mathcal{A}$ outputs its guess β' for β, $\mathcal{B}$ forwards this without change to the challenger. The advantage of $\mathcal{B}$ in winning the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{HIBE}}$ game is same as the advantage of $\mathcal{A}$ in winning the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$ game as all the responses to $\mathcal{A}$'s queries are simulated correctly by $\mathcal{B}$.

C Proof for Theorem 5.1

Theorem C.1. *Any time-deniable signature scheme satisfying the deniability property as defined in definition 5.5, also satisfies the k-hop deniability property as defined in definition 5.6.*

We prove this by a hybrid argument. In each subsequent hybrid $\mathcal{H}$ we will slightly modify the security game until we get to one where the advantage in distinguishing is indisputably negligible because both inputs will be drawn from the same distribution.

Let $\mathcal{H}_0$ be the original k-hop security game and consider the sequence of hybrids $\mathcal{H}_1 \ldots \mathcal{H}_k$ that are defined as follows:

$\mathcal{H}_i =$ In each of these hybrids our goal will be to eliminate the dependency of $\mathsf{AltSign}^k$ on the message (m_{i-1}, t_{i-1}). $\mathcal{H}_i$ is identical to $\mathcal{H}_{i-1}$ except when $\beta = 1$, σ is generated as

$$\mathsf{AltSign}^{k-i}(vk, (m_i, t_i, \sigma_i), \{(m_{i+1}, t_{i+1}), \ldots, (m^*, t^*)\}) \tag{9}$$

where $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i, t_i)$.

In the discussion that follows let $\mathcal{H}_i$ denote the outcome of $\mathcal{H}_i$, that is the probability that $\mathcal{A}$ correctly guesses the bit β' chosen by the challenger. We let the notation $\mathcal{H}_{i-1} \approx \mathcal{H}_i$ mean there is negligible difference between the success of $\mathcal{A}$ in $\mathcal{H}_{i-1}$ and $\mathcal{H}_i$.

Claim 4. $\forall i \in [k]$, $\mathcal{H}_{i-1} \approx \mathcal{H}_i$

For the sake of contradiction, suppose this is not true and $\exists$ a non-negligible function $\rho(\lambda)$ so that $|\Pr(\mathcal{A}_\lambda \text{ outputs } \beta' \mid \mathcal{H}_i) - \Pr(\mathcal{A}_\lambda \text{ outputs } \beta' \mid \mathcal{H}_{i-1})| = \rho(\lambda)$. Without loss of generality, assume $\Pr(\mathcal{A}_\lambda \text{ outputs } \beta' \mid \mathcal{H}_i) > \Pr(\mathcal{A}_\lambda \text{ outputs } \beta' \mid \mathcal{H}_{i-1})$. We now use this difference in success probabilities and the adversary $\mathcal{A}_\lambda$ to construct an adversary $\mathcal{B}$ that has non-negligible advantage in $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$.

Description of $\mathcal{B}$

- $\mathcal{B}$ receives vk from its challenger and sends it to $\mathcal{A}$. It also initializes an empty table $\mathcal{T}$ and uniformly samples $\beta' \xleftarrow{\$} \{0,1\}$
- In the query phase, $\mathcal{B}$ responds to the sign queries from $\mathcal{A}$ using the $\mathsf{O}_{\mathsf{Sign}}$ oracle: it forwards (m,t) to $\mathsf{O}_{\mathsf{Sign}}$, receives (id, σ), and records $(\mathsf{id}, m, t, \sigma)$ in $\mathcal{T}$. It then returns (id, σ) to $\mathcal{A}_\lambda$.
- In the query phase, let a challenge query to O_{Ch} from $\mathcal{A}_\lambda$ be $\mathsf{id}, \{(m_j, t_j)\}_{j\in[k-1]}, m^*, t^*$.
  - Check if $\mathcal{T}$ has an identifier id. If not output $\perp$.
  - If $\beta' = 0$, query $\mathsf{O}_{\mathsf{Sign}}$ on m^*, t^* to get $(\mathsf{id}^*, \sigma^*)$ and return σ^* to $\mathcal{A}_\lambda$
  - Else if $\beta' = 1$ and $i = 1$, let $\mathsf{id}' = \mathsf{id}$. Otherwise query $\mathsf{O}_{\mathsf{Sign}}$ on (m_{i-1}, t_{i-1}) to get id', σ. $\mathcal{B}$ makes a query to its challenge oracle with id', m_i, t_i and receives σ_i.
  - Compute $\sigma^* \leftarrow \mathsf{AltSign}^{k-i}(vk, (m_i, t_i, \sigma_i),$ $\{(m_{i+1}, t_{i+1}), \ldots (m^*, t^*)\})$ and send σ^* to $\mathcal{A}_\lambda$

- Let b be the output of $\mathcal{A}_\lambda$. If $\beta' = b$, return 0. Else return 1.

Analysis

We now analyze $\mathcal{B}$'s success probability. In the discussion that follows, let β be the bit chosen by the challenger in the $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$ game.

$$\begin{aligned}
&\Pr(\mathcal{B} \text{ outputs } \beta) \\
&= \frac{1}{2}\Pr(\mathcal{B} \text{ outputs } \beta \mid \beta = 0) + \frac{1}{2}\Pr(\mathcal{B} \text{ outputs } \beta \mid \beta = 1) \\
&= \frac{1}{2}\Pr(\mathcal{A}_\lambda \text{ succeeds in } \mathcal{H}_i \mid \beta = 0) + \frac{1}{2}\big[1 - \Pr(\mathcal{A}_\lambda \text{ succeeds in } \mathcal{H}_{i-1} \mid \beta = 1)\big] \\
&= \frac{1}{2} + \frac{1}{2}\big[\Pr(\mathcal{A}_\lambda \text{ succeeds in } \mathcal{H}_i \mid \beta = 0) - \Pr(\mathcal{A}_\lambda \text{ succeeds in } \mathcal{H}_{i-1} \mid \beta = 1)\big] \\
&= \frac{1}{2} + \frac{1}{2}\rho(\lambda) \\
&\implies \mathbf{Adv}^{\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}}_{\mathcal{B}} = \frac{1}{2}\rho(\lambda)
\end{aligned}$$

This advantage is non-negligible because ρ is non-negligible. And this contradicts our assumption that our scheme is secure in the sense of definition 5.5. Our claim follows.

Claim 5. $\forall$ *distinguishers* $\mathcal{D}$*, the advantage of* $\mathcal{D}$ *in* $\mathcal{H}_k$ *is* 0

Recall that in $\mathcal{H}_k$ we have replaced $\mathsf{AltSign}^k$ with $\mathsf{AltSign}^{k-k}(vk, (m^*, t^*, \sigma), \{\}) = \mathsf{AltSign}^0(vk, (m^*, t^*, \sigma), \{\}) = \sigma$ where $\sigma \leftarrow \mathsf{Sign}(sk, m^*, t^*)$. Therefore in $\mathcal{H}_k$ when computing any challenge queries the resulting computed signatures are

$$\sigma^0 \leftarrow \mathsf{Sign}(sk, m^*, t^*) \tag{10}$$

$$\sigma^1 \leftarrow \mathsf{Sign}(sk, m^*, t^*) \tag{11}$$

The distributions of σ^0 and σ^1 are thus identical and trivially no distinguisher can have an advantage in distinguishing between them.

D Lewko's Prime-Order HIBE Scheme

This is a description of Lewko's[23] prime order translation for an unbounded HIBE scheme. This scheme performs some operations over vectors of n-dimensional space, similar to Lewko's work we describe the scheme for $n = 10$.

$\mathsf{Setup}(1^\lambda) \to (pk, msk)$: The setup algorithm takes as input the security parameter 1^λ. A bilinear group G of sufficiently large prime order p is selected, where the bilinear map is denoted by $e : G \times G \to G_T$. The random dual orthonormal bases required for the scheme are also sampled as part of this algorithm $(\mathbb{D}, \mathbb{D}^*) \xleftarrow{R} \text{Dual}\left(\mathbb{Z}_p^n\right)$. Let $\mathbb{D} = \{\vec{d_1}, \ldots, \vec{d_n}\}$ and $\mathbb{D}^* = \{\vec{d_1^*}, \ldots, \vec{d_n^*}\}$. It also chooses random exponents $\alpha_1, \alpha_2, \theta, \sigma, \gamma, \xi \in \mathbb{Z}_p$. The public parameters, which we describe as part of the public key are

$$pk = \left\{G, p, e(g,g)^{\alpha_1 \vec{d_1} \cdot \vec{d_1^*}}, e(g,g)^{\alpha_2 \vec{d_2} \cdot \vec{d_2^*}}, g^{\vec{d_1}}, \ldots, g^{\vec{d_6}}\right\}$$

and the master secret key is

$$msk = \begin{array}{l}\{G, p, \alpha_1, \alpha_2, g^{\vec{d_1^*}}, g^{\vec{d_2^*}}, g^{\gamma \overrightarrow{d_1^*}}, g^{\xi \vec{d_2^*}}, g^{\theta \vec{d_3^*}}, g^{\theta \vec{d_4^*}}, \\ g^{\sigma \vec{d_5^*}}, g^{\sigma \vec{d_6^*}}\}\end{array}$$

$\mathsf{KeyGen}(msk, (ID_1, \ldots, ID_j)) \to sk_{ID}$: The key generation algorithm samples random values $r_1^i, r_2^i \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in [j]$. It also samples random values $y_i \xleftarrow{\$} \mathbb{Z}_p$ and $w_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [j]$ under the constraints that $y_1 + y_2 + \cdots + y_j = \alpha_1$ and $w_1 + w_2 + \cdots + w_j = \alpha_2$. For each $i \in [j]$, it computes:

$$K_i := g^{y_i \overrightarrow{d_1^*} + w_i \overrightarrow{d_2^*} + r_1^i ID_i \theta \vec{d}_3^* - r_1^i \theta \vec{d}_4^* + r_2^i ID_i \sigma \vec{d}_5^* - r_2^i \sigma \overrightarrow{d_6^*}}$$

The secret key is output as:

$$sk_{ID} = \left\{ g^{\gamma \vec{d}_1^*}, g^{\xi \vec{d}_2^*}, g^{\theta \vec{d}_3^*}, g^{\theta \vec{d}_4^*}, g^{\sigma d_5^*}, g^{\sigma \vec{d}_6^*}, K_1, \ldots, K_j \right\}$$

$\mathsf{Delegate}(sk_{ID}, ID_{j+1}) \to sk_{ID|ID_{j+1}}$: The delegation algorithm samples random values $\omega_1^i, \omega_2^i \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in [j+1]$. It also samples random values $y_i', w_i' \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [j+1]$ subject to the constraint that $y_1' + \cdots y_{j+1}' = 0 = w_1' + \cdots w_{j+1}'$. Let $g^{\gamma \vec{d}_1^*}, g^{\xi \vec{d}_2^*}, g^{\theta \vec{d}_3^*}, g^{\theta \vec{d}_4^*}, g^{\sigma \vec{d}_5^*}, g^{\sigma \vec{d}_6^*}, K_1, \ldots, K_j$ denote the elements of sk_{ID}. The delegated key $sk_{ID|ID_{j+1}}$ is formed as follows:

$$\begin{aligned}
sk_{ID|ID_{j+1}} := & \left\{ g^{\gamma \vec{d}_1^*}, g^{\xi \vec{d}_2^*}, g^{\theta \vec{d}_3^*}, g^{\theta \vec{d}_4^*}, g^{\sigma \vec{d}_5^*}, g^{\sigma \vec{d}_6^*} \right. \\
& K_1 \cdot g^{y_1' \gamma \vec{d}_1^* + w_1' \xi \vec{d}_2^* + \omega_1^1 ID_1 \theta \vec{d}_3^* - \omega_1^1 \theta \vec{d}_4^* + \omega_2^1 ID_1 \sigma \vec{d}_5^* - \omega_2^1 \sigma \vec{d}_6^*} \\
& \ldots, K_j \cdot g^{y_j' \gamma \vec{d}_1^* + w_j' \xi \vec{d}_2^* + \omega_1^j ID_1 \theta \vec{d}_3^* - \omega_1^j \theta \vec{d}_4^* + \omega_2^j ID_1 \sigma \vec{d}_5^* - \omega_2^j \sigma \overrightarrow{d_6^*}} \\
& g^{y_{j+1}' \gamma \vec{d}_1^* + w_{j+1}' \xi \vec{d}_2^* + \omega_1^{j+1} ID_{j+1} \theta \vec{d}_3^*} \\
& \left. \cdot g^{-\omega_1^{j+1} \theta \vec{d}_4^* + \omega_2^{j+1} ID_{j+1} \sigma \vec{d}_5^* - \omega_2^{j+1} \sigma \vec{d}_6^*} \right.
\end{aligned}$$

$\mathsf{Encrypt}(pk, M, ID) \to \text{ct}$: The encryption algorithm samples random values t_1^i, t_2^i for each $i \in [j]$, as well as random values $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_p$. It computes

$$C_0 := M e(g,g)^{\alpha_1 s_1 \vec{d}_1 \cdot \vec{d}_1^*} e(g,g)^{\alpha_2 s_2 \vec{d}_2 \cdot \overrightarrow{d_2^*}}$$

and

$$C_i := g^{s_1 \vec{d}_1 + s_2 \vec{d}_2 + t_1^i \vec{d}_3 + ID_i t_1^i \vec{d}_4 + t_2^i \vec{d}_5 + ID_i t_2^i \vec{d}_6}$$

for each $i \in [j]$. The ciphertext is $ct := \{C_0, C_1, \ldots, C_j\}$

$\mathsf{Decrypt}(ct, sk_{ID}) \to M$: The decryption algorithm computes

$$B := \prod_{i=1}^{j} e\left(C_i, K_i\right)$$

and computes the message as:

$$M = C_0 / B$$

D.1 Proof of Key-Indistinguishability for HIBE scheme of [23]

Theorem D.1. *The prime order variant of Lewko's scheme from [23] satisfies the key-indistinguishability property.*

The delegation algorithm (in Appendix D) in Lewko's prime order scheme [23] rerandomizes each exponent in a secret key. Each group element (the ones which are unique for an identity) in the key generated by $\mathsf{KeyGen}()$, is of the form:

$$K_i := g^{y_i \overrightarrow{d_1^*} + w_i \overrightarrow{d_2^*} + r_1^i ID_i \theta \vec{d}_3^* - r_1^i \theta \vec{d}_4^* + r_2^i ID_i \sigma \vec{d}_5^* - r_2^i \sigma \overrightarrow{d_6^*}}$$

Where the values $y_i \xleftarrow{\$} \mathbb{Z}_p$ and $w_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [j+1]$ are under the constraints that $y_1+y_2+\cdots+y_{j+1} = \alpha_1$ and $w_1 + w_2 + \cdots + w_{j+1} = \alpha_2$.

Whereas, in the key generated by $\mathsf{Delegate}()$, each group element (the ones which are unique for an identity) is of the form:

$$K'_i = K^*_i \cdot g^{y'_i\gamma\vec{d_1^*}+w'_i\xi\vec{d_2^*}+\omega_1^i ID_1\theta\vec{d_3^*}-\omega_1^i\theta\vec{d_4^*}+\omega_2^i ID_1\sigma\vec{d_5^*}-\omega_2^i\sigma\vec{d_6^*}}$$

Where K^*_i is the i'th group element of the key of the identity which was used as an input for $\mathsf{Delegate}()$. The following variables are sampled uniformly at random, $\omega_1^i, \omega_2^i \xleftarrow{\$} \mathbb{Z}_p$ for each $i \in [j+1]$. It also samples random values $y'_i, w'_i \xleftarrow{\$} \mathbb{Z}_p$ for $i \in [j+1]$ subject to the constraint that $y'_1 + \cdots y'_{j+1} = 0 = w'_1 + \cdots w'_{j+1}$.

The key fact to note is that the exponent of g in K'_i the variables $y_1^* + \gamma y'_1, \ldots, y_j^* + \gamma y'_j, \gamma y'_{j+1}$ are *randomly distributed* up to the constraint that their sum is α_1, and similarly $w_1 + \xi w'_1, \ldots, w_j + \xi w'_j, \xi w'_{j+1}$ are *randomly distributed* up to the constraint that their sum is α_2. Also, $r_1^i + \omega_1^i$ and $r_2^i + \omega_2^i$ are uniformly random for each i. The keys generated via $\mathsf{KeyGen}()$ are also sampled from the same distributions with the exact same constraints. This gives us the fact that the distribution of a secret key obtained through any sequence of delegations is the same as the distribution of a secret key for the same identity generated via $\mathsf{KeyGen}()$ making them statistically indistinguishable. In fact, this is noted by Lewko in the description of the scheme as well. Moreover, the fact that the adversary has the master secret key msk, doesn't give it any advantage because the two keys generated only differ in the randomness used to generate them, having the msk doesn't give the adversary any way to distinguish between these two because they are statistically indistinguishable. Therefore, the challenge key pairs $(sk_\beta, sk_{1-\beta})$ for the HIBE key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{HIBE}}$ are indistinguishable to any PPT adversary.

E On the Necessity of Time-Lock Encryption

Definition E.1 (Time-Lock Encryption)**.** A time-lock encryption scheme for computational reference clock $\mathcal{C}(1^\lambda)$, consists of the following two PPT algorithms:

- $\mathsf{TL.Encrypt}(1^\lambda, m, \tau) \to ct$: The encryption algorithm takes as input the message, a security parameter, and an integer to represent time.
- $\mathsf{TL.Decrypt}(ct, w) \to m/\perp$: The decryption algorithm takes as input a ciphertext, and a witness and outputs either a message m or the symbol $\perp$.

It has the following **correctness** property:

$$\Pr\left[\begin{array}{c} c \xleftarrow{\$} \mathsf{TL.Encrypt}\left(1^\lambda, \tau_{\mathrm{dec}}, m\right); \\ w_\tau \xleftarrow{\$} \mathcal{C}\left(1^\lambda, \tau\right); \\ m' := \mathsf{TL} \cdot \mathsf{Decrypt}\left(w_\tau, c\right) : \\ m = m' \end{array}\right] = 1$$

Security. We say an *admissible* polysize adversaries $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}} = \{(\mathcal{A}_{\lambda,0}, \mathcal{A}_{\lambda,1})\}_{\lambda\in\mathbb{N}}$, $\mathcal{A}_\lambda = (\mathcal{A}_0, \mathcal{A}_1)$, (t, τ, ε)-breaks time-lock encryption scheme $(\mathsf{TL.Encrypt}, \mathsf{TL.Decrypt})$ for a clock $\mathcal{C}$, if $\mathrm{size}(\mathcal{A}) \in \mathrm{poly}(\lambda)$ and $\tau \in \mathbb{N}$ such that

$$\Pr\left[\begin{array}{c} (m_0, m_1, st) \xleftarrow{\$} \mathcal{A}_0^{\mathcal{C}}\left(1^\lambda, \tau\right); \quad b \xleftarrow{\$} \{0,1\}; \\ c \leftarrow \mathsf{TL.Encrypt}(\left(1^\lambda, \tau, m_b\right); b' \leftarrow \mathcal{A}_1^{\mathcal{C}}\left(1^\lambda, c, st\right) : \\ b = b' \end{array}\right] - 1/2 > \varepsilon$$

Where admissibility of an adversary is defined similar to subsection 5.1.

Witness encryption schemes allow encrypting a message m to an instance x of an NP language and allows decryption using a valid witness w such that $(x, w) \in R$.

Definition E.2 (Extractable Witness Encryption)**.** A witness encryption scheme for an NP relation R consists of the following polynomial time algorithms:

- $\mathsf{WE.Encrypt}(1^\lambda, x, m) \to ct$: This algorithm takes as input the security parameter 1^λ, an unbounded string x which represents an instance of R and a message m. It outputs a ciphertext ct which encrypts message m.
- $\mathsf{WE.Decrypt}(ct, w) \to m$: This algorithm takes as input a ciphertext ct and a witness w and outputs a message m or the symbol $\perp$.

An extractable WE scheme has the following properties:

- **Correctness.** For all $(x, w) \in R$ and every message m:

$$\Pr\big[\mathsf{WE.Decrypt}(\mathsf{WE.Encrypt}(1^\lambda, x, m), w) = m\big] = 1 - \mathsf{negl}(\lambda)$$

- **Extractable Security.** Let ($\mathsf{WE.Encrypt}$, $\mathsf{WE.Decrypt}$) be a witness encryption scheme for an NP relation R, such a scheme is secure if for all PPT adversaries $\mathcal{A}$ and all polynomials q, there exists a PPT extractor E and a polynomial p, such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$:

$$\Pr\left[\begin{array}{l} b \xleftarrow{\$} \{0,1\}; ct \leftarrow \mathsf{WE.Encrypt}(1^\lambda, x, m) : \\ \mathcal{A}(x, ct, z) = m \end{array}\right] \geq 1/2 + 1/q(|x|)$$

$$\implies \Pr\left[\ E(x, z) = w : (x, w) \in R\ \right] \geq 1/p(|x|)$$

Time-Lock Encryption from Time-Deniable Signatures. To construct a time-lock encryption scheme using deniable signatures and extractable witness encryption, consider $(vk, sk) \leftarrow \mathsf{DS.KeyGen}(1^\lambda, T(\lambda))$ and $\sigma \leftarrow \mathsf{DS.Sign}(sk, m, t)$. Now consider a witness encryption scheme which encrypts to statements of the form $x = (m, t, \sigma, vk)$ for a relation R where for witnesses of the forms $w = (m^*, t^*, \sigma^*)$, $(x, w) \in R$ if $\mathsf{DS.Verify}(vk, \sigma^*, m^*, t^*) = 1$. We also provide the intuition behind why this scheme is secure. The time-lock encryption algorithms proceed as follows

1. $\mathsf{TL.Encrypt}(1^\lambda, m, t + T)$: It outputs $ct \leftarrow \mathsf{WE.Encrypt}(1^\lambda, x, m)$, where $x = (m, t, \sigma, vk)$.
2. $\mathsf{TL.Decrypt}(ct, w)$: Since it takes time T to create σ^* from σ, after time T a valid witness is available to run the decryption algorithm for WE with witness (m^*, t^*, σ^*). Output $m' \leftarrow \mathsf{WE.Decrypt}(ct, w)$.

The intuition behind the security argument is essentially that no admissible adversary should be able to distinguish an encryption of m_0 from an encryption of m_1 as this adversary is depth bounded. Otherwise, such an adversary computes $w \in R$, i.e., a different signature σ^* on some message, timestamp pair (m^*, t^*) by performing significantly less operations than the number of operations required. This adversary is solving the time-lock puzzle in sequential time less than T. Given such a distinguishing adversary we can leverage the extractor for witness encryption to break the unforgeability property for deniable signatures.

F On the Necessity of Secure Timestamps

Recall that in our definition, the $\mathsf{AltSign}$ algorithm takes as input a previously computed valid signature (or forgery). In particular, our notion does not rely upon the use of cryptographic timestamps.

An alternative notion discussed in Section 2 is one where $\mathsf{AltSign}$ does not require a previously computed signature as input; instead it only uses a timestamp issued by an external server to create a forgery. We

argue that in the latter case, the timestamps issued by the server must be cryptographic (and in particular, unpredictable or unforgeable, depending on the implementation).

Suppose this is not the case. Then we can devise a simple attack using the $\mathsf{AltSign}$ algorithm to break the unforgeability of the signature scheme. Consider a (non-uniform) adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ that wants to generate a forged signature for *any* message m^*, and *any* time-stamp t^*. Since we allow for arbitrary polynomial time pre-processing in the unforegeability game, $\mathcal{A}_0$ runs $\mathsf{AltSign}$ on input m^* and $g(t^*)$ to compute a forged signature, where $g(\cdot)$ computes the output of the time server for time t^* (this also captures the scenario that the time stamp is entirely ignored by $\mathsf{Sign}/\mathsf{AltSign}$). Since there is no security property associated with the timestamps issued by the server, $g(\cdot)$ is a function that can be computed efficiently, so $\mathcal{A}_0$ is polynomial time.

Let σ^* be the forged signature computed by $\mathcal{A}_0$, who passes it along to $\mathcal{A}_1$ to output as its forgery. Since the above strategy works for *any* (m^*, t^*), and $\mathcal{A}_1$ needs only a single computational step (to output the forged signature received from $\mathcal{A}_0$), this attack constitutes a valid forgery of the time-deniable signature scheme.

G Epochal Signatures

We recall the unforgeability game and definitions from the work [21] by Hülsing and Weber. An epochal signature scheme ES has the following four algorithms:

$\mathsf{ES.KeyGen}(1^\lambda, \Delta t, E, V) \rightarrow (pk, sk)$: Takes as input a security-parameter 1^λ, an epoch-length Δt, the maximum number of epochs $E \in \mathsf{poly}(\lambda)$ and the number of epochs $V < E$ for which the signatures are valid and generates the long-term key pair (pk, sk).

$\mathsf{ES.Evolve}(sk) \rightarrow (pinfo_e, sk'/\perp)$: Takes as input the long-term secret key sk and returns the public epoch information $pinfo_e$ and an updated secret key sk' or $\perp$ if sk has already been evolved E times.

$\mathsf{ES.Sign}(sk_e, m) \rightarrow \sigma$: Takes as input a secret key sk_e and a message m and outputs a signature σ for the corresponding epoch e.

$\mathsf{ES.Verify}(pk, e, \sigma, m) \rightarrow b$: Takes as input the public key pk, an epoch e, a signature σ and a message m and returns a bit b.

The unforgeability definition is defined with respect to the unforgeability game $\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}$ defined as follows:

Setup. The challenger runs $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta t, E, V)$ and gives the public key pk to the adversary $\mathcal{A}$. It sets t_0 to the current time, sets $e = 0$ and initializes the set of queries $q = [\emptyset, \ldots, \emptyset]$, where the k'th entry in the set corresponds to the set of queries asked in epoch k.

Query Phase. In this phase, the adversary gets to query two different oracles.

1. The key evolution oracle $\mathsf{O}_{\mathsf{Evolve}}(\cdot)$ takes as input some wall clock time t, checks that $t \geq t_0 + e.\Delta t$ which indicates that the current time is the e'th epoch. It computes $(pinfo_e, sk_e) \leftarrow \mathsf{Evolve}(sk)$ if t satisfies the above properties. At the end of an $\mathsf{O}_{\mathsf{Evolve}}(sk, \cdot)$ query, the adversary also gets access to the corresponding sign oracle for epoch e, $\mathsf{O}_{\mathsf{Sign}}(sk_e, \cdot)$.
2. The sign oracle $\mathsf{O}_{\mathsf{Sign}}(sk_e, \cdot, \cdot)$ takes as input a secret key sk_e, wall-clock time t and a message m. If $t < t_0 + e.\Delta t$, it outputs a signature $\sigma \leftarrow \mathsf{Sign}(sk_e, m)$ on the message and updates the corresponding epoch in the query set $q[e] \cup \{m\}$.

Forge. The adversary outputs its forgery (σ', m') and wins (game outputs 1) if:

1. For the corresponding epoch e', there is no query corresponding to this message $(m', e') \notin q[e']$.
2. $\mathsf{Verify}(pk, e', \sigma', m')$ outputs 1.

The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda, \Delta t) = \Pr\Big[\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}(1^\lambda, \Delta t, E, V) = 1\Big]$.

Remark: To the best of our knowledge the authors do not define the function $\mathsf{now}()$. We assume that this is the current time value and hence implies the existence of a wall clock.

Definition G.1 (Unforgeability of Epochal Signatures)**.** An epochal signature scheme Σ is unforgeable if there is no efficient adversary that has a non-negligible chance of winning the unforgeability game $\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}$:

$$\forall \mathcal{A} \in \text{ PPT}, \lambda \in \mathbb{N}, E \in \mathsf{poly}(\lambda), V \in \{1, \ldots, E-1\} :$$

$$\mathbf{Adv}_{\mathcal{A}}(1^\lambda, \Delta t) \leq \mathsf{negl}(\lambda)$$

Because the time-lock puzzle security definition in their work is different from ours, we give the indistinguishability game below and denote it as $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{TL}}$. It is assumed in this game that the challenger has access to a wall-clock.

Setup. The challenger sends the security parameter 1^λ and the difficulty parameter Δt to the adversary.

Challenge. The adversary $\mathcal{A}$ picks the challenge messages (m_0, m_1) and sends them to the challenger. The challenger performs the following operations:

- Picks a random bit $b \xleftarrow{\$} \{0, 1\}$.
- Computes $c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t, m_b)$, sends c to $\mathcal{A}$ and sets current time as t_0.

Response. $\mathcal{A}$ sends its guess b' for b to the challenger and wins (game outputs 1) if the following conditions are satisfied:

- $b = b'$.
- Let the wall-clock time for when the challenger receives $\mathcal{A}$'s guess be t_1, then it should be true that $t_1 - t_0 < \Delta t$.

The advantage of the adversary $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}(1^\lambda, \Delta t) = \Pr\Big[\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{TL}}(1^\lambda, \Delta t) = 1\Big]$.

Definition G.2 ($\mathsf{TimeLock}$ security according to [21])**.** A time-lock puzzle is secure if $\forall \mathcal{A} \in \mathsf{poly}(\lambda)$, $\mathbf{Adv}_{\mathcal{A}}(1^\lambda, \Delta t) \leq \mathsf{negl}(\lambda)$.

G.1 Faulty Epochal Signature Construction

Given a secure epochal signature construction Σ one can construct another secure epochal signature scheme Σ' which has undesirable properties as discussed in 3. We give a formal description and then briefly argue why the scheme is still a secure epochal signature scheme while failing to meet the time-deniable signature security definition. We only describe those algorithms with significant changes from Σ and only use these deviations for $V > 1$ (for $V = 1$, we use all algorithms from Σ).

$\Sigma'.\mathsf{Evolve}(\mathsf{sk}_e) \rightarrow \mathsf{sk}_{e+1}$: this is a randomized evolve where $sk_e = (sk'_e, r)$ where sk'_e is the signing key output by Σ and $r = r_1||r_2||r_3||r_4$ is a pad, where $\forall i, r_i$ are chosen uniformly. Let $sk'_{e+1} = \Sigma.\mathsf{Evolve}(sk'_e)$. If e is even, $r' = r$. If r is odd, then we sample r' as above. The output is (sk'_{e+1}, r').

$\Sigma'.\mathsf{Sign}((sk'_e, r_1||r_2||r_3||r_4), m) \rightarrow \sigma'$: First, construct the signature $\sigma = \mathsf{Sign}(sk'_e, m)$. Then compute $c_{sk} = sk_e \oplus r_3 \oplus r_4$. If e is an even epoch then $c_{TL} = \mathsf{TimeLock}(1^\lambda, \Delta t + \epsilon, r_1)$, otherwise we put r_2 in the time lock puzzle. If e is even and $m = r_1$, define $z = r_3$. And if e is odd $m = r_2$ then $z = r_4$. Otherwise z is a random pad. The final output signature is $\sigma, c_{sk}, c_{TL}, z$.

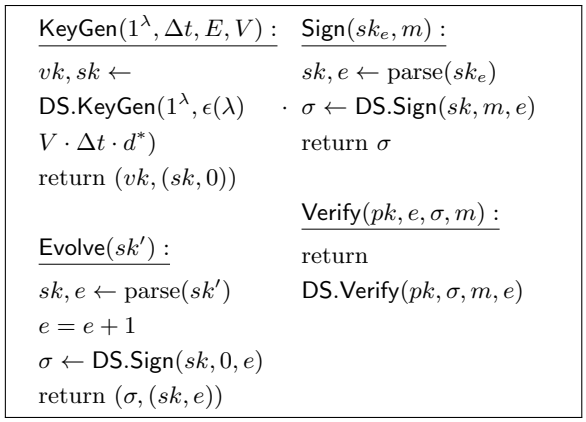

Figure 5: An ES construction from a time-deniable signature scheme DS. $\epsilon(\lambda)$ is the admissibility parameter for the time-deniable signature scheme. The sentinel message for $pinfo$ is $m = 0$.

In words, our construction utilizes a one time pad to encrypt the signing key and outputs this ciphertext as part of the signature along with a time lock puzzle of "target" queries r_1 or r_2 and what is (normally) a random pad. In the epochal signature unforgeability game the secret key is inaccessible to the adversary despite having c_{sk} because it is unlikely they will ask sign queries on r_1 and r_2 before they can break the time lock puzzle, at which point they can no longer ask the relevant queries because the Δt time window has passed. Deniability still holds because these r values are ephemeral and we only structure c_{sk} in this way for $V \neq 1$, meaning that the judge $\mathcal{J}$ never sees r_3 or r_4 for e_0 as part of sk. This construction is not unforgeable in ous definition because the pre-processing adversary is not bound by the Δt time window and can thus recover sk_e.

G.2 Time-Deniable Signatures as Epochal Signatures

In this section we show any secure DS scheme can be generically transformed into a secure ES scheme. At a high level, our construction is a simple transformation where verification and signing uses the time-deniable signature scheme and the evolve algorithm keeps track of the current epoch. $pinfo_e$ contains a signature on a dummy, sentinel message at the timestamp corresponding to epoch e. Because of the different models of time considered by the two primitives, we do a translation between wall-clock time and circuit depth.We make the following assumption: for any circuit C that terminates in wall-clock time t, the depth of C when it terminates is $d_C \cdot t$ where d_C is a constant that depends on C. Let C be the set of circuits for which some input x causes C to terminate before or at wall-clock time t and let $d^* = \max_{C \in \mathsf{C}} d_C$. d^* is needed to correctly set the time parameter given to the time-deniable signature scheme's KeyGen algorithm. The construction appears in Figure 5.

Theorem G.1. *The* ES *scheme presented in Figure 5 is unforgeable if the time-deniable signature scheme* DS *is unforgeable.*

We prove this by contradiction, supposing there exists an adversary $\mathcal{B}$ who succeeds with non-negligible advantage in the ES unforgeability game and then using that adversary to construct an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ for the DS unforgeability game. For the ES unforgeability definition we assume that before the counter e is given to the Verify algorithm it is appropriately advanced.

Description of $\mathcal{A}$:

- Receive vk from the challenger. Give vk to $\mathcal{B}$. Initialize a counter $ctr = 0$ and the wall clock time $t_{init} = \mathsf{now}()$.

- On queries $\mathsf{O}_{\mathsf{Evolve}}$ check if $t' \geq t_{init} + (e+1)\Delta t$ where t' is the current wall clock time. If yes, set $ctr = ctr + 1$ and query $\mathsf{O}_{\mathsf{Sign}}$ on message 0 to receive σ. Return σ as $pinfo$. Else do not advance the counter and output $\perp$.
- On queries $\mathsf{O}_{\mathsf{Sign}}$ from $\mathcal{B}$ with message m, query the challenger's $\mathsf{O}_{\mathsf{Sign}}$ oracle with m and timestamp ctr. Return σ to $\mathcal{B}$.
- If $\mathcal{B}$ returns signature σ^* in epoch e^* on message m^*, then $\mathcal{A}$ returns (m^*, e^*, σ^*) as its forgery.

First, we argue that $\mathcal{A}$ is an admissible adversary in the time-deniable signature unforgeability game. Let $\mathcal{A}_0$ denote the interactions of $\mathcal{A}$ with $\mathcal{B}$ before epoch e^* begins. Because for an ES scheme, $\mathsf{size}(\mathcal{B}) \in \mathsf{poly}(\lambda)$ and since $\mathcal{A}$ is also doing $\mathsf{poly}(\lambda)$ work while interacting with $\mathcal{B}$, we have that $\mathsf{size}(\mathcal{A}_0) \in \mathsf{poly}(\lambda)$. If we let the output of $\mathcal{A}$ and $\mathcal{B}$ after this interaction be an advice string z, we can then split off the rest of $\mathcal{A}$ and $\mathcal{B}$'s interaction as $\mathcal{A}_1$. For $\mathcal{B}$ to be a successful adversary, they must be able to produce m^*, σ^* before wall clock time $V\Delta t$ has past since the start of epoch e^*. Then we have an upper bound on the circuit depth of $\mathcal{B}$ from the start of e^* until termination as $d^*V\Delta t$. Since $\mathcal{A}$ just forwards queries between the challenger and $\mathcal{B}$ the overhead it adds is minimal (on the order of the number of queries made by $\mathcal{B}$) and can be ignored for the sake of this proof sketch. $\mathsf{depth}(\mathcal{A}_1)$ is therefore appropriately bounded as $d^*\Delta t \cdot V \leq \frac{d^*\Delta t \cdot V \cdot \epsilon(\lambda)}{\epsilon(\lambda)}$.

We now argue that if $\mathcal{B}$ is successful in its forgery so is $\mathcal{A}$. As said earlier, in order for $\mathcal{B}$ to succeed it must produce a valid pair (m^*, σ^*) before the wall clock time bound where validity means that $\mathsf{DS.Verify}$ succeeds given the current timestamp is e^* and that $\mathcal{B}$ has never asked for a signature on m^* at time e^*. The tuple (m^*, e^*, σ^*) is thus a valid forgery for $\mathcal{A}$ as well.

Theorem G.2. *The* ES *scheme presented in Figure 5 is deniable if the time-deniable signature scheme* DS *is deniable.*

Suppose this is not true and there exists a judge $\mathcal{J}$ that succeeds with non-negligible advantage in the ES deniability game. Then we will construct an adversary $\mathcal{A}$ that succeeds with non-negligible advantage in the DS deniability game.

Description of $\mathcal{A}$:

- Receive vk, sk from the challenger. Forward sk to $\mathcal{J}$.
- Uniformly sample a random message m. When $\mathcal{J}$ specifies its challenge (m^*, e_0, e_1), query $\mathsf{O}_{\mathsf{Sign}}$ with $(m, e_0 + e_1)$ and receive (id, σ).
- Query O_{Ch} with id, m^*, e_0 to get σ^*. Send σ^* to $\mathcal{J}$. If $\mathcal{J}$ responds with b send b to the challenger.

$\mathcal{J}$ expects to see one of two signatures. One creates the signature by evolving sk e_0 times while the other evolves the key $e_0 + e_1$ times and uses $pinfo_{e_0+e_1}$. When the challenger's bit $b = 0$, $\mathcal{A}$ produces the output of $\mathsf{ES.Sign}$ in Figure 5 which is simply $\mathsf{DS.Sign}$. When $b = 1$, the output is produced by the simulator $\mathcal{S}$ in the ES deniability game which is the equivalent to $\mathsf{DS.AltSign}$ in our construction. Therefore, the distributions $\mathcal{J}$ sees are correct and if $\mathcal{J}$ is successful in distinguishing then so is $\mathcal{A}$.