Time-Deniable Signatures

Gabrielle Beck
Johns Hopkins University
Baltimore, MD, USA
becgabri@cs.jhu.edu

Arka Rai Choudhuri
NTT Research
Sunnyvale, CA, USA
arkarai.choudhuri@ntt-research.com

Matthew Green
Johns Hopkins University
Baltimore, MD, USA
mgreen@cs.jhu.edu

Abhishek Jain
Johns Hopkins University
Baltimore, MD, USA
abhishek@cs.jhu.edu

Pratyush Ranjan Tiwari
Johns Hopkins University
Baltimore, MD, USA
pratyush@cs.jhu.edu

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, at least *for a sender-chosen period of time*. Once this time period has elapsed, however, time-deniable signatures can be forged by any party who obtains a signature. This forgery property ensures that signatures serve a useful authentication purpose for a bounded time period, while also allowing signers to plausibly disavow the creation of older signed content. Most critically, and unlike many past proposals for deniable authentication, TDS do not require interaction with the receiver or the deployment of any persistent cryptographic infrastructure or services beyond the signing process (*e.g.,* APIs to publish secrets or author timestamp certificates.)

We first investigate the security definitions for time-deniability, demonstrating that past definition attempts are insufficient (and indeed, allow for broken signature schemes.) We then propose an efficient construction of TDS based on well-studied assumptions.

KEYWORDS

Digital Signatures, Deniability

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 [12] 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 goals 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 [31, 38, 41]: this is possible because DKIM signing keys are long-lived, and the protocol's non-repudiable signatures can be verified long after an email has been received and processed. 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 to facilitate such verification. Since email signing is implemented by commercial mail providers rather than end-users, users of popular services cannot opt-out. These developments have ignited a technical debate around the desirability of long-term non-repudiability guarantees in widely-used protocols such as email [23], 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 [1, 8, 42], 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 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 can create deliberate *forgeries* after the necessary anti-spam and spoofing checks have been completed. The two protocols employ different 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

[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 [12].

Proceedings on Privacy Enhancing Technologies 2023(3), 79–102

https://doi.org/10.56553/popets-2023-0071

has elapsed. Others have made even simpler proposals wherein DKIM providers simply rotate and publish existing DKIM signing keys on a periodic basis [11, 23]. Each proposal seeks to build signatures that are unforgeable for a period of time necessary to support short-term transport checks, but become forgeable after this period.

The major limitation of the proposals above is that forgery requires the active cooperation of signers, or else depends on the continuous operation of new trusted infrastructure such as "time servers" that publish keys or timestamp certificates on a periodic basis [40]. 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 often-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*, with only the requirement 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 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 AltSign algorithm requires the forger to perform a pre-specified number of *sequential* operations δ, where 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 therefore require that forgeries are indistinguishable from signatures produced using the ordinary signing algorithm, and in fact that even *linking* forgeries to the specific signatures that were used to create them should be challenging. This indistinguishability property is a fundamentally novel property of this work, that is not present in previous attempts to solve this problem [3, 24]. It also has important follow-on implications: since forgeries are indistinguishable from true signatures, this implies that any forgery must be useful 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 signature carries a 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 a timestamp greater than $t - t_\delta$ where t_δ is the minimal expected time needed to compute a forgery.[2] This approach requires only that honest senders and receivers possess loosely synchronized clocks.

Our contributions. In this work we investigate the problem of building time-deniable signatures. We first develop formal definitions for this new primitive, then present a construction 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: while developing our definitions, we found that previous efforts to formalize the security of deniable authentication schemes fall short. For example, we show that the security definitions for some related primitives [24] contain subtle weaknesses that admit practically-insecure constructions. To provide evidence for the robustness 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 [24] 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 can be realized using standard assumptions in pairing-based cryptography and sequential puzzles based on repeated-squaring assumption [36], without the need for zkSNARKs or other heavy-weight constructions.

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

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 AltSign, that *does not require* the signing key sk to generate signatures. Intuitively, as long as the signatures generated by Sign and 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 for formalizing the above intuition and defining time-deniable signatures.

Challenge I: Preventing pre-computation of forgeries. Recall that any party can compute a forgery (via the algorithm AltSign) after the unforgeability period expires. But how do we ensure that a party cannot execute 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 [17, 34]. For example, when signing a message 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 TimeForge scheme of Specter *et al.* [40] and a recent proposal by Bonneau *et al.* [3].

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 was authored 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.[3]

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 arbitrarily 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 strong clock synchronization. The closely related prior work of *epochal signatures* by Hülsing and Weber [24] considers a security notion that crucially relies on various participants having synchronized clocks. Roughly, in an 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 state that no adversary can forge signatures for an epoch during the epoch. As we show in §3, the security definitions for epochal signatures are *fragile*: there exist epochal signature schemes that are secure under the given definitions and yet become *completely insecure when clocks are even slightly out of sync*. This problem stems from the fact that the unforgeability notion proposed for the primitive puts strict time limits on the adversary while it queries a signing oracle. We show that if enforcement of these query restrictions is violated (even slightly) by a real-world signing oracle at epoch e, an epochal signature scheme can become catastrophically insecure *for all future epochs*.

Unfortunately, avoiding such outcomes is not easy, and in this work, we seek to strengthen our security definitions to avoid such issues. We do this in two ways: unlike [24], our definitions model the unforgeability period computationally – through the widely-adopted technique of bounding the number of sequential computation steps the adversary may compute [6, 16, 35, 36, 43]. While this still requires conversion when used in the real world, it does not embed the conversion into the security definition. Much 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. During this phase, the adversary is given free rein (within only a polynomial time-bound) to query the signing oracle and forge signatures. This phase significantly loosens the restrictions on the adversary, allowing them to query for signatures and run the AltSign algorithm (or any other process) as many times as they wish. Once the pre-processing phase is complete, the adversary then enters a second *forgery* phase in which their runtime is more strictly bounded. Our sole restriction is that the forgery produced in the second phase must be computed on a timestamp t^* that is greater than any timestamp queried during the pre-processing phase.

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 KeyGen, Sign, AltSign and Verify. The KeyGen and Verify algorithms are the same

[3] 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 J.

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 the correctness of the scheme, we require that AltSign generates a verifying signature as long as it's 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.

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 a time stamp $> t^*$.

Deniability. This property asks an adversary to distinguish between a "fresh" signature generated using Sign, and a signature generated using 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 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* 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 to sign any message m' with time stamp $t' \leq t$. Such a key can then be used by the 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* [36]. 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 [24], 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) [4, 5, 9]. 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:

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

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 $t \leq T$.[4]

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) *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 a 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 to 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 potentially 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.

[4] 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.

Intuitively, we prove unforgeability by leveraging the unforgeability of the functional signature scheme and the security of time-lock puzzles, 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 an 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 [7] - 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[5], resulting in at most $\log(t)$ many keys. A detailed description is provided in the technical sections, but here we provide an illustrative example.

In the HIBE identity tree of Figure 1, the key corresponding to f_{10} is both sk_0 and sk_{10}. To derive a key for f_{00}, one executes the HIBE's delegate algorithm using sk_0 to create the key sk_{00}. In fact, to derive a key for $\mathsf{sk}_{f_{t'}}$ from sk_{f_t} for *any* $t' \leq t$ one can simply use the HIBE delegation algorithm, i.e. there is no need to run the key generation algorithm afresh.

Looking ahead, 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 [26].

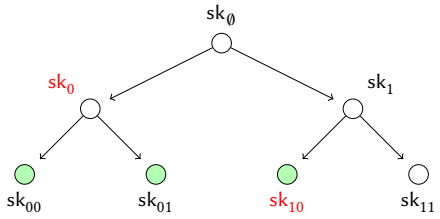

Figure 1: Each node in the tree represents a HIBE secret key $\mathsf{sk}_{\mathsf{id}}$ for the identity id. $\mathsf{Trace}(root = \emptyset, 10)$ constitutes the nodes which represent the secret key for f_{10}, i.e. $\mathsf{sk}_{f_{10}} = (\mathsf{sk}_0, \mathsf{sk}_{10})$. Using this set, all messages with prefixes in the green nodes can be signed.

3 RELATED WORK

Concurrent work. A concurrent and independent work of Arun et al. [3] 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 afterward. However, their work considers a different model 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. Our work also has many similarities to that of [40]. Particularly our construction is similar to one of theirs in its usage of a HIBE for creating signatures. However, their setting is more limited: they assume that a central server provides key material for forgery so that if the server is knocked offline, deniability does not necessarily hold. For us, the ability to forge solely depends on seeing the signature itself. This change in the model comes with new subtle challenges in the indistinguishability of forgeries and signatures and in formulating security definitions that account for an adversary who has access to a polynomial time forgery algorithm.

Epochal Signatures. Our work is closely related to the prior work on epochal signatures [24]. At a very high level, epochal signatures aim to achieve deniability in a manner similar to ours - 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).[6] The constructed epochal signature in [24] 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

[5] 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.

[6] 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.

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 [24] is limited to be *bounded use*. The term *bounded* here means polynomially-bounded. To be specific, the number of epochs that their system can support is bounded ahead of time by some polynomial in the security parameter. This is an outcome of the run-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* the system is initialized, and once the total number of epochs is surpassed, the entire system needs to be reset from scratch. If a system must be reset too often, and resetting is costly (i.e. involves running an expensive key generation algorithm), it may limit the usability of the system. The broad question of *bounded* vs *unbounded* use is not new and has been studied in various contexts in cryptography such as bounded vs unbounded query chosen-ciphertext secure encryption [14], depth-bounded vs depth-unbounded hierarchical identity-based encryption [27] and homomorphic encryption [19], bounded-collusion vs unbounded collusion in functional encryption [22, 37], 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 construction to have a significant asymptotic improvement over [24] that may translate to concrete practical costs for some large parameter sets.

Need for Clock Synchronization. As discussed earlier, the unforgeability notion in [24] requires the participants to have perfectly synchronized clocks. We now demonstrate that if such a requirement is not met, then the consequences can be catastrophic and result in a compromise of security for all future epochs. Specifically, we construct a secure epochal signature scheme where the unforgeability property can be broken when the clocks are slightly out of sync. 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 an over-simplified presentation of our counter-example to convey the general idea. The full counter-example is more involved (due to technical reasons) and is presented in Appendix K.1.

Intuitively, we exploit the restricted signing oracle in the unforgeability definition of epochal signatures which prevents an adversary from receiving signatures in any epoch e outside of a fixed real time window of size Δt. Our epochal signature scheme makes use of a special *trigger* message m_e^* which differs per epoch. If the adversary queries for a signature on message m_e^* in epoch e, then they receive some "secret information" from the signing oracle which can be used to recover the signing key. If the message space is large enough and m_e^* is chosen uniformly at random, this modification would not make our scheme insecure, as an adversary would only have a negligible chance of guessing m_e^*. We therefore modify the signing oracle so that, in addition to handing out signatures on messages m for epoch e, it time-lock puzzle encrypts m_e^* with difficulty parameter Δ_t' where $\Delta t < \Delta t' < \Delta t + \varepsilon$. 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". In our actual counter-example, this secret information cannot directly be equal to the secret key because the ES scheme must be perfectly deniable even to someone that holds the original signing key. To deal with this, we instead encrypt the signing key with a one time pad that is 2 out of 2 additively secret shared. Querying on different trigger messages reveals different shares of the key. Further details can be found in Appendix K.1.

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 secret key material is used to key the signing key, this is not revealed as a part of pinfo_e and does not need to be to create indistinguishable signatures (every field of the signature will be simulatable). Thus, simply having pinfo_e will not allow recovery of sk.

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 attack described above by decrypting the time-lock puzzles, making the relevant queries, and using the results to obtain the signing key. The key is then passed on to the "online adversary" who uses it to produce a forged signature. We remark that, again, the above description is oversimplified and the full counter-example is presented in K.1.

We briefly summarize some of the different properties of proposed constructions for expiring signature schemes in Table 1.

4 PRELIMINARIES

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. A function f has sequential time d or takes d sequential steps if for all circuits C that correctly compute f the smallest circuits C have $\mathsf{depth}(C) = d$. 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 [36]. We now briefly give a formal description of a time lock puzzle.

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.

	ES [24]	Short-Lived Sigs [3]	KeyForge B [40]	Us
Forging Assumptions	None	RB	Publish Keys	None
Max number epochs	Bounded	Unbounded	Unbounded	Unbounded
Forgers all derive same key	Yes	Yes	Yes	No
Building Blocks	OWF + TLP	tVDF + RB*	BDH	BDH + TLP

Table 1: Comparison of a subset of existing constructions that provide a notion of deniability for signatures. tVDF stands for a trapdoor VDF, while RB is a randomness beacon.

$\mathsf{Gen}(1^\lambda, \Delta, s) \to Z$: 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 $\mathsf{size}(\mathcal{A}_\lambda) \in \mathsf{poly}(\lambda)$ and depth $\mathsf{depth}(\mathcal{A}_\lambda) \leq \frac{T(\lambda)}{\alpha(\lambda)}$, $\exists$ a negligible function μ s.t.$\forall \lambda \in \mathbb{N}$, $\forall s_0, s_1 \in \{0,1\}^\lambda$,

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

This is a variation of the time lock puzzle definition of [15], 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 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 many HIBE schemes can be modified to have this property [20, 26, 27].

4.1.1 Security. The notion of security we consider for HIBE is the adaptive variant defined in works by Lewko and Waters [27, 29]. For a full description of the security game see Appendix A.

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 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, except the adversary 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 $\mathcal{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$

$\mathcal{QK}(\cdot)$ 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).

$\mathcal{O}_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ 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 4.2 (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 can be easily satisfied by many existing HIBE schemes, provided the sub-key components from earlier levels of the HIBE can be re-randomized. Randomization techniques like these have been used to construct anonymous HIBEs in the past [39]. In Appendix G, we show that the prime order variant of the Lewko-Waters HIBE scheme [26] satisfies this property.

5 TIME-DENIABLE SIGNATURES: DEFINITION

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

- $\mathsf{KeyGen}(1^\lambda, T = T(\lambda)) \to (vk, sk)$: On input the security parameter 1^λ and a difficulty parameter for AltSign called T, this randomized algorithm outputs the verification key vk and the signing key sk.
- $\mathsf{Sign}(sk, m, t) \to \sigma$: On input a message m and the signing key sk, this randomized algorithm outputs a signature σ on m for timestamp t.
- $\mathsf{Verify}(vk, \sigma, m, t) \to \{0, 1\}$: On input a signature σ, a message m, verification key vk and timestamp t, this deterministic algorithm outputs a bit.
- $\mathsf{AltSign}\ (vk, (m^*, t^*, \sigma^*), m, t) \to \sigma$: 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 KeyGen, Sign, Verify must run in time poly in the size of the input. For AltSign it is required that there exist a positive polynomial q such that $\forall T = T(\lambda), \forall \lambda \in \mathbb{N}$, 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:

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

where $\mathsf{AltSign}^0(vk, (m_0, t_0, \sigma_0), \{\}) = \sigma_0$. In words, $\mathsf{AltSign}^k$ is a signature obtained by applying 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 $j \in [k]$:

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T);\\ \sigma_0 \leftarrow \mathsf{Sign}(sk, m_0, t_0);\\ \sigma \leftarrow \mathsf{AltSign}^k(vk, (m_0, t_0, \sigma_0),\\ \{(m_j, t_j)\}_{j\in[k]}) :\\ \mathsf{Verify}(vk, \sigma, m_k, t_k) = 1\end{array}\right] = 1 \tag{2}$$

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 the 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.

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 $\mathcal{O}_{\mathsf{Sign}}$ which is defined as follows. On input a message m and a timestamp t, the signing oracle $\mathcal{O}_{\mathsf{Sign}}(sk, \cdot, \cdot)$ 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 $\mathcal{O}_{\mathsf{Sign}}$.
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) $\forall i, t^* > t_i$ for queries (m_i, t_i) made by $\mathcal{A}_0$
(2) $\forall i, (m_i, t_i) \neq (m^*, t^*)$ where (m_i, t_i) are queries made to $\mathcal{O}_{\mathsf{Sign}}$ 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 $\forall \lambda \in \mathbb{N}, \epsilon(\lambda) \leq T(\lambda)$, $\mathsf{depth}(\mathcal{A}_1) \leq \frac{T(\lambda)}{\epsilon(\lambda)}$, and $\mathsf{size}(\mathcal{A}) \in \mathsf{poly}(\lambda)$. 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 5.3 ((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 $\lambda \in \mathbb{N}$:

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, T(\lambda)),\\ (z) \leftarrow \mathcal{A}_{\lambda,0}{}^{\mathcal{O}_{\mathsf{Sign}}(sk,\cdot,\cdot)}(vk),\\ (m^*, t^*, \sigma^*) \leftarrow \mathcal{A}_{\lambda,1}{}^{\mathcal{O}_{\mathsf{Sign}}(sk,\cdot,\cdot)}(z) :\\ \mathsf{Verify}(vk, m^*, t^*, \sigma^*) = 1\end{array}\right] \leq \mu(\lambda) \tag{3}$$

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 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 attackers.

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 $O_{\mathsf{Sign}}(sk,\cdot,\cdot)$ and a challenge oracle $O_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$.

$O_{\mathsf{Sign}}(sk,\cdot,\cdot)$ 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 (id,m,t,σ) in $\mathcal{T}$, and returns (id,σ)

$O_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ 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 5.4 (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 AltSign algorithm k times instead of just once. Intuitively, this notion could be stronger and offer more deniability via a larger pool of indistinguishable 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 $O_{\mathsf{Sign}}(\mathsf{sk},\cdot,\cdot)$ and $O_{\mathrm{Ch}}(\cdot,\cdot,\cdot,\cdot)$. O_{Sign} is the same oracle as given in $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$.

$O_{\mathrm{Ch}}(\cdot,\cdot,\cdot,\cdot)$ 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 5.5 (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 5.6. *Any time-deniable signature scheme satisfying the deniability property as defined in definition 5.4, also satisfies the k-hop deniability property as defined in definition 5.5.*

For a proof of Theorem 5.6 see Appendix F.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[10].

Definition 6.1. A functional signature scheme FS = (Setup, KeyGen, Sign, 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 and Security. Security for a functional signature scheme is the traditional notion of unforgeability where the adversary is given access to the verification key vk. For completeness, correctness and the full security definition is included in Appendix B

6.1 Key Delegation

In order to create signing keys even without the master signing key, we define an additional PPT algorithm called 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 $f : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\perp\}$, 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.

$\mathsf{FS.Delegate}(mvk,f,sk_f,f') \rightarrow sk_{f'},\perp$: 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 $f : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\perp\}$ supported by FS, for all restrictions f' of f, and $\forall m \in \mathcal{X}$ where $f'(m) \neq \perp$:

$$\Pr\left[\begin{array}{l}(mvk,msk) \leftarrow \mathsf{FS.Setup}(1^\lambda);\\ sk_f \leftarrow \mathsf{FS.KeyGen}(msk,f);\\ sk_{f'} \leftarrow \mathsf{FS.Delegate}(mvk,f,sk_f,f');\\ \sigma \leftarrow \mathsf{FS.Sign}(sk_{f'},f',m) :\\ \mathsf{FS.Verify}(mvk,f'(m),\sigma) = 1\end{array}\right] = 1 \qquad (4)$$

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 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 $O_{\mathsf{Key}}(\cdot)$, 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 $O_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ 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

$$f_T(t, m) = \begin{cases} t||m & t \le T \\ \perp & \text{otherwise} \end{cases} \tag{5}$$

For the sake of readability, in the following construction 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.

We now give a description of how to build the delegatable FS for the function f_T described previously. Given a HIBE scheme, we consider identities I of the form $\{0,1\}^m$ for $m \le l$. These identities can be described by a binary tree of depth l that has 2^l leaves where every HIBE identity is either an interior node or a leaf. The identity corresponding to a given node n is defined by the path $path_n$ from the root to n: i.e. let $p_0 \ldots p_{m-1}, p_m = n$ be the nodes along the path to n (including n) then the identity is $b_1 \ldots b_m$ where b_i is 0 if p_i is the left child of p_{i-1} and 1 otherwise. Signing a message m for time t will correspond to extracting a key for the identity $t\|m$ from the leaf node t. To generate a prefix key for f_T where $T < 2^l$, for every node p_t in $path_T$ besides T itself, we extract a signing key for the left child of p_t. These keys, along with an extracted key for T itself, make up the functional key. For correctness, we note that to sign for any time $M < T$ we can derive the leaf node M if we have a key for an ancestor of M in the tree, and for every $M < T$ it has an ancestor that is a left child of some node p_t along $path_T$. Delegation works for a similar reason. We note that for the purposes of key indistinguishability if the intersection of $path_T$ and $path_M$ contains some node n, the key associated with n's left child from $path_T$ must be re-randomized in f_M. Verification of a message m simply checks the included HIBE key for the identity $t||m$ by attempting to encrypt and decrypt a random message, as is suggested in [7].

The construction is presented in pseudocode in Figure 2.

Theorem 6.4. *If* HIBE *is adaptively secure then the functional signature scheme for prefix functions constructed in Figure 2 is unforgeable.*

For a proof of Theorem 6.4 see Appendix E.

Theorem 6.5. *If* HIBE *is key-indistinguishable then the scheme in Figure 2 satisfies the functional signatures key-indistinguishability property.*

For a proof of Theorem 6.5 see Appendix E.2.

7 CONSTRUCTION OF TIME-DENIABLE SIGNATURES

This section describes our construction of time-deniable signatures from key indistinguishable, delegatable FS for prefix functions and time lock encryption. To sign a message at timestamp 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 time-lock encrypted to produce a ciphertext that is sent along with the signature. The alternate signing algorithm decrypts the ciphertext, uses the delegate algorithm to produce an appropriate signing key for $f_{t'}$ with $t' \le t$, and then signs the message and time-lock encrypts the signing key. For the 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 |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.

For proofs of unforgeability and deniability see Appendix D.

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

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

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

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

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

$\underline{\mathsf{KeyGen}(1^\lambda, T):}$
$(mvk, msk) \leftarrow \mathsf{FS.Setup}(1^\lambda)$
return $((mvk, T, \lambda), (msk, T, \lambda))$

$\underline{\mathsf{Sign}(sk = (msk, T, \lambda), m, t):}$
$sk_t \leftarrow \mathsf{FS.KeyGen}(msk, f_t)$
$v, s \leftarrow \mathsf{FS.Sign}(sk_t, f_t, (t, m))$
$c \leftarrow \mathsf{TimeLock.Gen}(1^\lambda,$
$T \cdot z(\lambda), sk_t)$
return (c, s)

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

$\underline{\mathsf{Verify}(vk, \sigma, m, t):}$
$c, s = \mathsf{parse}(\sigma)$
return $\mathsf{FS.Verify}(vk, t||m, s)$

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 $\bar{t} \le t$, 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.

8 SYSTEM INTEGRATION

We now give a high-level description of a system that could utilize time-deniable signatures: electronic mail. We first define the two main actors in any signature scheme.

Signer: The signer is a party that publishes messages that can later be authenticated. In the setting of email, this is usually a domain owner that sets up a DKIM record to sign outgoing mail e.g. Google. Instead of using a regular signature scheme, they would run the TDS.Sign algorithm using the mail as the message the timestamp they are signing the message at, and a signing key sk produced by the TDS.KeyGen algorithm.

Verifier: The verifier is a party receiving the message and looking to verify its authenticity. In our setting, this would be a mail server accepting inbound mail and attempting to verify that the message is from the claimed domain. Using DNS, the mail server pulls the relevant key for verification. In this case, the key is a TDS verification key, and this key is used to run the TDS.Verify algorithm. The server would also check when the message was signed and the time parameter Δ to determine if the message is too stale to check for authenticity.

Based on this they would decide whether to forward mail to users or not.

Note that the algorithm TDS.AltSign never needs to be run by any party. Just the *existence* of the algorithm itself is enough to cast doubt on any message sent longer ago than $now() - \Delta$.

We now consider two different adversaries which are common in such a setting.

Forger: The forger is a party who sees multiple messages and attempts to construct one that verifies without having access to a signing key. In our setting, a forger gets to see old keys which would allow them to sign messages "from the past", but these messages have already expired authenticity and cannot be verified. Therefore, we consider forgers who are trying to sign messages for the current signing window or into the future. In the email setting, this could be a small-scale adversary such as a random hacker spear-phishing someone, or a more well-equipped adversary like a nation-state-funded attacker.

Detective: The detective is a party tries to discover whether or not some message from the past was sent by the signer or a different party. The message is guaranteed to have been sufficiently far in the past that Δ has already passed. In the email scenario, this is equivalent to a reporter who discovers emails - perhaps through a leak - and tries to verify whether they came from the claimed domain. The detective is not a one-shot adversary and may get access to multiple signed messages over time.

We now make some remarks on the forger and detective. Both detective and forger may induce the signer to sign messages of their choosing. The forger may do some pre-computation work before attempting to attack a scheme, but once the forger decides to attack they must find a validating signature before the time window expires. The detective is a long-lived adversary who may even recover the entire signing key in the future. Even in this scenario, it should not be possible for the detective to distinguish a true signature from a forged one. Our one requirement is that they cannot see the message before its time period has expired, or gain access to a proof that the message existed starting at some time period. This problem appears to be inherent for all schemes aiming to achieve similar properties to time-deniability.

9 IMPLEMENTATION AND EVALUATION

Implementation. To demonstrate the efficiency of our scheme, we implemented it in python. For our time lock puzzle, we modified an existing, open-source implementation of an RSW time-lock puzzle [25]. Our timestamp supports 2^{16} different values which is approximately equivalent to what is supported by [40]. This is reasonable given at least some motivating applications (i.e. email), where frequent key rotation is done for domains and coarse granularity may be acceptable.

Construction of FS. To instantiate our functional signature scheme, we need a HIBE that is both key-indistinguishable and adaptively secure. We consider two different HIBE schemes: one a variant of the Unbounded HIBE from Lewko and Waters [28] due to Lewko [26, 28], the other a HIBE from Chen et al. [13]. Both schemes are adaptively secure and have tight reductions. We prove they satisfy the key-indistinguishability property of Section 4.1.2 in Appendix G.1. For Lewko's scheme, we tweaked an existing python implementation [33] and instantiated using the curve SS512. For the scheme of Chen et al, we modify an existing IBE implementation from the same paper in the Charm library [2] and instantiate with BN254. Different curves are necessary since Lewko's construction only works with symmetric pairings whereas Chen et al use asymmetric ones. We hereafter call these constructions L−SS512 and CLLWW−BN254. The curve SS512 natively offers ≈ 80 bits of security while BN254 offers ≈ 110 bits.

Setting Parameters. There are two main concerns that come with implementing time-based crypto assumptions: one is capturing the speed-up offered by parallelism, the other is accurately estimating the fastest real-world time to do the computational task the assumption is based on. On the first point, to the best of our knowledge, there are no known improvements from bounded parallelism against the RSW assumption. For the second, recent results [32, 44] suggest that an FPGA implementation can achieve $\approx 2^{24}$ squarings per second and an ASIC $\approx 2^{28}$ squares per second. For our implementation below, we benchmarked the cost of computing squares modulo a 2048-bit composite on our machine. This corresponded to roughly 5,883,206 squares per second which is a factor of 4 less than the FPGA cost reported above.

Experimental Evaluation. Experiments were done on an Intel Xeon E5 with 500GB of memory, running Ubuntu. Our implementation uses neither multi-threading or multiprocessing. Estimates were obtained by running each algorithm 500 times and taking the median. For the rest of this section, let N denote the arity of the tree and d be the depth. The timestamp value in our experiments is chosen uniformly at random per each run, as signing time and signature size differ significantly depending on the value of t.

We begin the analysis by examining the effect of varying N. It is an important parameter for our scheme because N and d must satisfy $N^d > 2^{16}$ and together determine the efficiency of signing and verifying. To be explicit, signing consists of extracting at most $d \cdot (N-1)$ keys from the HIBE tree where each key is $O(d)$ group elements long and requires $O(d)$ work to generate. Thus $O(Nd^2)$ work must be done in signing where $d = \lceil \log_N(2^{16}) \rceil$. This quantity is minimized when N is close to 7 meaning that signing time and signature size are optimal when $N = 7$ as can be seen in Figure 4 and Appendix H respectively. Although we do not depict it, larger values of N *always* result in a decrease in verification time since verification depends only on d. Our microbenchmarks are presented in Table 2 for $N = 7$. The superiority of the signing algorithm in CLLWW−BN254 to L−SS512 can be attributed to the use of asymmetric over symmetric pairings and because in L−SS512 each level of the HIBE adds ten group elements to the HIBE key whereas in CLLWW−BN254 it only adds four. Because signing mostly consists of creating these keys, it heavily impacts performance and the size of the signature itself.

10 CONCLUSION

In this work we introduced a new notion of deniable signatures that provides strong unforgeability and deniability guarantees without requiring the signer to periodically publish secret key material. We show how to realize our primitive using time lock puzzles

Scheme	KeyGen(ms)	Verify(ms)
L−SS512	360	200
CLLWW−BN254	433	619
Scheme	Sign(ms)	Sig. Size (B)
L−SS512	2695	417090
CLLWW−BN254	542	77424

Table 2: Microbenchmarks for the scheme of Figure 3 with $N = 7, d = 6$ and using L−SS512 and CLLWW−BN254.

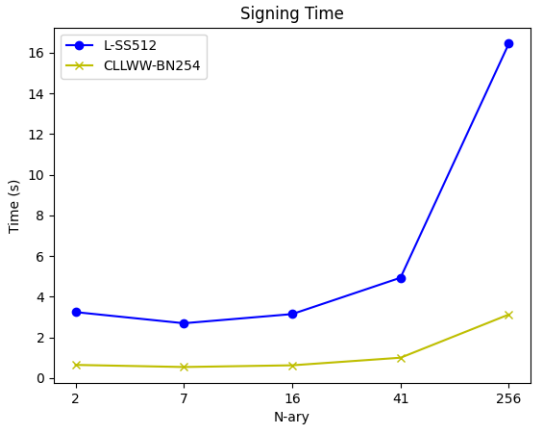

Figure 4: The signing time for a TDS using L−SS512 and CLLWW−BN254, varying values of N

and a HIBE scheme that satisfies a special key-indistinguishability property. Important directions for future work include constructing time-deniable signatures from a different set of assumptions (non-HIBE based) and building constructions that satisfy robust correctness.

ACKNOWLEDGMENTS

GB and MG are supported by DARPA under Contract No. HR001120C0084. MG additionally is supported by CNS-1653110, CNS- 1801479, DARPA under Contract No. HR00112020021, and a Google Security & Privacy Award. A part of this research was done when AC was at UC Berkeley, where they were at the time supported in part by DARPA under Agreement No. HR00112020026, AFOSR Award FA9550-19-1-0200, NSF CNS Award 1936826, and research grants by the Sloan Foundation, and Visa Inc. AJ is supported by NSF CNS-1814919, NSF CAREER 1942789, a Johns Hopkins University Catalyst award, AFOSR Award FA9550-19-1-0200, Office of Naval Research Grant N00014-19-1-2294 and research gifts from Ethereum, Stellar and Cisco. PRT was supported in part by Office of Naval Research under awards N00014-19-1-2294 and N00014-19-1-2292 and by the NSF under awards CNS-1814919 and CNS-1653110. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

REFERENCES

[1] 2013-2022. Signal Protocol Documentation. https://www.signal.org/docs/.
[2] Joseph A Akinyele, Christina Garman, Ian Miers, Matthew W Pagano, Michael Rushanan, Matthew Green, and Aviel D Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering* (2013).
[3] Arasu Arun, Joseph Bonneau, and Jeremy Clark. 2022. Short-lived zero-knowledge proofs and signatures. , 190 pages.
[4] Michael Backes, Sebastian Meiser, and Dominique Schröder. 2016. Delegatable Functional Signatures. In *PKC 2016, Part I (LNCS, Vol. 9614)*, Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang (Eds.). Springer, Heidelberg, 357–386. https://doi.org/10.1007/978-3-662-49384-7_14
[5] Mihir Bellare and Georg Fuchsbauer. 2014. Policy-Based Signatures. In *PKC 2014 (LNCS, Vol. 8383)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 520–537. https://doi.org/10.1007/978-3-642-54631-0_30
[6] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable Delay Functions. In *CRYPTO 2018, Part I (LNCS, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, 757–788. https://doi.org/10.1007/978-3-319-96884-1_25
[7] Dan Boneh and Matthew K. Franklin. 2001. Identity-Based Encryption from the Weil Pairing. In *CRYPTO 2001 (LNCS, Vol. 2139)*, Joe Kilian (Ed.). Springer, Heidelberg, 213–229. https://doi.org/10.1007/3-540-44647-8_13
[8] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. 2004. Off-the-record communication, or, why not to use PGP. In *WPES 2004*.
[9] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 2014. Functional Signatures and Pseudorandom Functions. In *PKC 2014 (LNCS, Vol. 8383)*, Hugo Krawczyk (Ed.). Springer, Heidelberg, 501–519. https://doi.org/10.1007/978-3-642-54631-0_29
[10] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. 2014. Functional signatures and pseudorandom functions. In *International workshop on public key cryptography*. Springer, 501–519.
[11] Jon Callas. 2020. Re: [Ietf-dkim] DKIM key rotation best practice. https://mailarchive.ietf.org/arch/msg/ietf-dkim/uxjvOlaE9Ho3KyfYVEZDxBgVJVE/
[12] Jon Callas, Eric Allman, Miles Libbey, Michael Thomas, Mark Delany, and Jim Fenton. 2010. DomainKeys Identified Mail (DKIM) Signatures.
[13] Jie Chen, Hoon Wei Lim, San Ling, Huaxiong Wang, and Hoeteck Wee. 2013. Shorter IBE and Signatures via Asymmetric Pairings. In *PAIRING 2012 (LNCS, Vol. 7708)*, Michel Abdalla and Tanja Lange (Eds.). Springer, Heidelberg, 122–140. https://doi.org/10.1007/978-3-642-36334-4_8
[14] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, abhi shelat, and Vinod Vaikuntanathan. 2007. Bounded CCA2-Secure Encryption. In *ASIACRYPT 2007 (LNCS, Vol. 4833)*, Kaoru Kurosawa (Ed.). Springer, Heidelberg, 502–518. https://doi.org/10.1007/978-3-540-76900-2_31
[15] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. 2020. Non-Malleable Time-Lock Puzzles and Applications. *IACR Cryptol. ePrint Arch.* 2020 (2020), 779.
[16] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. 2020. SPARKs: Succinct Parallelizable Arguments of Knowledge. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Heidelberg, 707–737. https://doi.org/10.1007/978-3-030-45721-1_25
[17] Ethereum Foundation . 2022. The Ethereum Beacon Chain. Available at https://ethereum.org/en/eth2/beacon-chain/.
[18] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. 2013. Witness encryption and its applications. In *45th ACM STOC*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM Press, 467–476. https://doi.org/10.1145/2488608.2488667
[19] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *41st ACM STOC*, Michael Mitzenmacher (Ed.). ACM Press, 169–178. https://doi.org/10.1145/1536414.1536440
[20] Craig Gentry and Alice Silverberg. 2002. Hierarchical ID-Based Cryptography. In *Advances in Cryptology — ASIACRYPT 2002*, Yuliang Zheng (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 548–566.
[21] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. 2013. How to Run Turing Machines on Encrypted Data. In *CRYPTO 2013, Part II (LNCS, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Heidelberg, 536–553. https://doi.org/10.1007/978-3-642-40084-1_30
[22] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. 2012. Functional Encryption with Bounded Collusions via Multi-party Computation. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Heidelberg, 162–179. https://doi.org/10.1007/978-3-642-32009-5_11
[23] Matthew D. Green. 2020. OK Google: Please publish your DKIM secret keys. https://blog.cryptographyengineering.com/2020/11/16/ok-google-please-publish-your-dkim-secret-keys/
[24] Andreas Hülsing and Florian Weber. 2021. Epochal Signatures for Deniable Group Chats. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. 1677–1695.
[25] Jonathan Levi. 2020. time-lock-puzzle. Available at https://github.com/drummerjolev/time-lock-puzzle.

[26] Allison B. Lewko. 2012. Tools for Simulating Features of Composite Order Bilinear Groups in the Prime Order Setting. In *EUROCRYPT 2012 (LNCS, Vol. 7237)*, David Pointcheval and Thomas Johansson (Eds.). Springer, Heidelberg, 318–335. https://doi.org/10.1007/978-3-642-29011-4_20
[27] Allison B. Lewko and Brent Waters. 2011. 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*. 547–567.
[28] Allison B. Lewko and Brent Waters. 2011. Unbounded HIBE and Attribute-Based Encryption. In *EUROCRYPT 2011 (LNCS, Vol. 6632)*, Kenneth G. Paterson (Ed.). Springer, Heidelberg, 547–567. https://doi.org/10.1007/978-3-642-20465-4_30
[29] Allison B. Lewko and Brent Waters. 2014. Why Proving HIBE Systems Secure Is Difficult. In *EUROCRYPT 2014 (LNCS, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, Heidelberg, 58–76. https://doi.org/10.1007/978-3-642-55220-5_4
[30] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. 2018. How to build time-lock encryption. *Des. Codes Cryptogr.* 86, 11 (2018), 2549–2586.
[31] Jeremy B. Merrill. 2017. Authenticating Email Using DKIM and ARC, or How We Analyzed the Kasowitz Emails.
[32] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. 2022. Low-Latency ASIC Algorithms of Modular Squaring of Large Integers for VDF Evaluation. *IEEE Trans. Comput.* 71, 1 (2022), 107–120. https://doi.org/10.1109/TC.2020.3043400
[33] Nikos Fotiou. 2014. HIBE-LW11. Available at https://github.com/nikosft/HIBE_LW11.
[34] NIST. 2019. NIST Randomness Beacon (prototype implementation). Available at https://beacon.nist.gov/home.
[35] Krzysztof Pietrzak. 2019. Proofs of Catalytic Space. In *ITCS 2019*, Avrim Blum (Ed.), Vol. 124. LIPIcs, 59:1–59:25. https://doi.org/10.4230/LIPIcs.ITCS.2019.59
[36] Ronald L Rivest, Adi Shamir, and David A Wagner. 1996. Time-lock puzzles and timed-release crypto. (1996).
[37] Amit Sahai and Brent R. Waters. 2005. Fuzzy Identity-Based Encryption. In *EUROCRYPT 2005 (LNCS, Vol. 3494)*, Ronald Cramer (Ed.). Springer, Heidelberg, 457–473. https://doi.org/10.1007/11426639_27
[38] Raphael Satter. 2018. Emails: Lawyer who met Trump Jr. tied to Russian officials.
[39] Elaine Shi and Brent Waters. 2008. Delegating capabilities in predicate encryption systems. In *International Colloquium on Automata, Languages, and Programming*. Springer, 560–578.
[40] Michael A. Specter, Sunoo Park, and Matthew Green. 2021. KeyForge: Non-Attributable Email from Forward-Forgeable Signatures. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. 1755–1773.
[41] Craig Timberg, Matt Viser, and Tom Hamburger. 2022. Here's how The Post analyzed Hunter Biden's laptop.
[42] Nik Unger and Ian Goldberg. 2018. Improved Strongly Deniable Authenticated Key Exchanges for Secure Messaging. *Proc. Priv. Enhancing Technol.* 2018, 1 (2018), 21–66.
[43] Benjamin Wesolowski. 2019. Efficient Verifiable Delay Functions. In *EUROCRYPT 2019, Part III (LNCS, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, 379–407. https://doi.org/10.1007/978-3-030-17659-4_13
[44] Erdinç Öztürk. 2020. Design and Implementation of a Low-Latency Modular Multiplication Algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 6 (2020), 1902–1911. https://doi.org/10.1109/TCSI.2020.2966755

A HIBE SECURITY

We consider HIBE security similarly to the work of Lewko and Waters[27, 29] 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 A.1 (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.

B FUNCTIONAL SIGNATURES

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,

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

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 $\bar{O}_{\mathsf{Key}}$ and a signing oracle $\bar{O}_{\mathsf{Sign}}$.

(1) $\bar{O}_{\mathsf{Key}}(msk, \cdot, \cdot)$ 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^i_f. Otherwise generate $sk_f \leftarrow \mathsf{KeyGen}(msk, f)$, record (i, f, sk_f) in $\mathcal{T}$ and return sk_f.
(2) $\bar{O}_{\mathsf{Sign}}(msk, \cdot, \cdot, \cdot)$ 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 $\bar{O}_{\mathsf{Sign}}$, $m_i \neq m^*$

(2) there is no key sk_f queried from $\bar{O}_{\mathsf{Key}}$ 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

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

C 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 [18, 21], we show that time-deniable signatures imply time-lock puzzles.

We demonstrate this implication in Appendix I. We remark that while extractable witness encryption is a strong tool, it alone is not known to imply time-lock puzzles.[7]

D PROOFS FOR TIME DENIABLE SIGNATURES

Theorem D.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 O_{Sign} by $\mathcal{A}_{\lambda,1}$ is not r, output $\perp$.

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

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,

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

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 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* $|\mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_i}(\lambda) - \mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_{i-1}}(\lambda)| \leq \mathbf{Adv}_{\mathcal{B}}^{\epsilon-\mathsf{TimeLock}}(\lambda)$

[7] When supplemented with a computational reference clock, it is known to imply time lock puzzles [30].

WLOG, assume $\mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_i} > \mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_{i-1}}$. 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 $\mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_i}(\lambda) - \mathbf{Adv}_{\mathcal{A}}^{\mathcal{H}_{i-1}}(\lambda)$ 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 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 |A.B| denote the depth bound on the algorithm B for primitive A. The depth bound for $\mathcal{B}$ is represented below

$$\mathsf{depth}(\mathcal{B}) \leq \mathsf{depth}(\mathcal{A}_1) + m \cdot \Big(|\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}| + |\mathsf{FS.Sign}|\Big) + |\mathsf{FS.Verify}| + 2 \leq \frac{t(\lambda)}{\epsilon(\lambda)}$$

$$+m \cdot \Big(|\mathsf{FS.KeyGen}| + |\mathsf{TimeLock.Gen}| + |\mathsf{FS.Sign}|\Big) + |\mathsf{FS.Verify}| + 2$$

$$\leq \frac{\epsilon(\lambda)(|\mathsf{FS.Verify}|+2)}{\epsilon(\lambda)} + \frac{t(\lambda)[1+|\mathsf{FS.KeyGen}|+|\mathsf{TimeLock.Gen}|+|\mathsf{FS.Sign}|]}{\epsilon(\lambda)} \leq \frac{t(\lambda)z(\lambda)}{\epsilon(\lambda)}$$

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

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

$\mathcal{B}$'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. $\exists C$, *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 C:

- Receive mvk from the FS challenger.
- On queries m, t to $O_{\mathsf{Sign}}(\cdot)$
  - if phase one, query $\bar{O}_{\mathsf{Key}}(f_t, i)$ 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 $\bar{O}_{\mathsf{Sign}}(f_t, i, (t, m))$ 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 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 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 C a signature that 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 C returns a valid forgery and the claim follows.

Theorem D.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 $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{DS}}$ game to construct an adversary $\mathcal{B}$ which wins the delegatable functional signatures key-indistinguishability game $\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{FS}}$.

- $\mathcal{B}$ 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 $O_{\mathsf{Sign}}(sk, \cdot, \cdot)$ query on message m and timestamp t, $\mathcal{B}$ performs the following operations:
  - Query the FS key oracle $\bar{O}_{\mathsf{Key}}(msk, \cdot, \cdot)$ 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 $O_{\mathsf{Sign}}(sk, t, m)$ 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 $O_{\mathsf{Ch}}(\cdot, \cdot, \cdot)$ 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 DS.Sign() algorithm when the challenger's sampled bit β is 0 as the key used for FS.Sign() is a freshly generated key. When $\beta = 1$, the key used for FS.Sign() is a delegated key, which simulates the 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 $\mathcal{B}$.

E PROOFS FOR DELEGATABLE FUNCTIONAL SIGNATURES

Theorem E.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 $\bar{O}_{\mathsf{Key}}$ 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 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 $\bar{O}_{\mathsf{Sign}}$ 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 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 E.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 $\bar{O}_{\mathsf{Key}}(\cdot)$ 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 $O_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ 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 $O_{\mathrm{Ch}}(\cdot,\cdot,\cdot)$ 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 FS.KeyGen() or 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 $\mathcal{B}$.

F PROOF OF K-HOP DENIABILITY

Theorem F.1. *Any time-deniable signature scheme satisfying the deniability property as defined in definition 5.4, also satisfies the k-hop deniability property as defined in definition 5.5.*

We prove this by a hybrid argument, starting with the security game where the challenger chooses $\beta = 1$ and ending with one where they choose $\beta = 0$. Because the advantage of $\mathcal{A}$ will change negligibly between hybrids, we will be able to say that the difference between the output of the adversary when $\beta = 0$ and $\beta = 1$ is negl. which is equivalent to adversarial advantage being negl. within a factor of 2. Where details are omitted in a description of hybrid $\mathcal{H}_i$ it is assumed they are the same as $\mathcal{H}_{i-1}$.

Let $\mathcal{H}_0$ be the k-hop security game with $\beta = 1$ and consider the sequence of hybrids $\mathcal{H}_1 \ldots \mathcal{H}_{k-1}$ that are defined as follows:

$\mathcal{H}_i$ = Set $\beta = 1$. Generate σ as
$$\mathsf{AltSign}^{k-i}(vk, (m_i, t_i, \sigma_i), \{(m_{i+1}, t_{i+1}), \ldots, (m^*, t^*)\}) \quad (6)$$
where $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i, t_i)$.

In the discussion that follows, let $\mathbf{Adv}^{\mathcal{H}_i}_{\mathcal{A}}(\lambda)$ be the advantage of $\mathcal{A}$ when it plays the modified game $\mathcal{H}_i$ and let $\mathbf{Adv}^{\mathcal{H}_i,\mathcal{H}_{i-1}}_{\mathcal{D}}(\lambda, \mathcal{A})$ denote the advantage of a distinguisher $\mathcal{D}$ in distinguishing $\mathcal{A}$'s advantage in $\mathcal{H}_i$ and $\mathcal{H}_{i-1}$

Claim 4. $\forall i \in [k], |\boldsymbol{Adv}^{\mathcal{H}_{i-1}}_{\mathcal{A}_\lambda}(\lambda) - \boldsymbol{Adv}^{\mathcal{H}_i}_{\mathcal{A}_\lambda}(\lambda)| \leq \mathsf{negl}(\lambda)$

Suppose this is not true. Then $\exists$ a distinguisher $\mathcal{D}$, s.t. $\mathbf{Adv}^{\mathcal{H}_i,\mathcal{H}_{i-1}}_{\mathcal{D}}(\lambda, \mathcal{A}_\lambda) = \rho(\lambda)$ and $\rho(\lambda)$ is non-negligible. We now use $\mathcal{D}$ 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 $\mathcal{A}_\lambda$. It also initializes an empty table $\mathcal{T}$.
- In the query phase, $\mathcal{B}$ responds to the sign queries from $\mathcal{A}_\lambda$ using the O_{Sign} oracle: it forwards (m, t) to O_{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 $i = 1$, let $\mathsf{id}' = \mathsf{id}$. Otherwise query O_{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 $\mathcal{A}_\lambda$. Send the result of the equality check $b == 1$ to $\mathcal{D}$. If $\mathcal{D}$ returns z return $\bar{z}$.

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.

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

This advantage is non-negligible because ρ is non-negligible. However, on the other hand, the advantage must be negligible because our scheme satisfies one-hop security. Thus no such distinguisher can exist and the claim follows.

$\mathcal{H}_{k+1}$ = set $\beta = 0$

Let X^0 be the distribution of the output of $\mathcal{A}$ in $\mathcal{H}_k$ and X^1 be the distribution of the output of $\mathcal{A}$ in $\mathcal{H}_{k+1}$.

Claim 5. $\forall$ *n.u.p.p.t distinguishers* $\mathcal{D}$,
$|\Pr(D(x)=1|x\leftarrow X^0)-\Pr(D(x)=1|x\leftarrow X^1)|\le \mathsf{negl}(\lambda)$

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, $\mathcal{H}_k$ and $\mathcal{H}_{k+1}$ result in $\mathcal{A}$ seeing the exact same distribution of input. This means the output of $\mathcal{A}$ cannot depend on β and $X^0 = X^1$. Thus the claim is trivially true.

G LEWKO'S PRIME-ORDER HIBE SCHEME

This is a description of Lewko's[26] 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 $(\mathbb{D},\mathbb{D}^*)\xleftarrow{R}\mathrm{Dual}\left(\mathbb{Z}_p^n\right)$. 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

$$msk=\{G,p,\alpha_1,\alpha_2,g^{\vec{d}_1^*},g^{\vec{d}_2^*},g^{\gamma\overrightarrow{d_1^*}},g^{\xi\vec{d}_2^*},g^{\theta\vec{d}_3^*},g^{\theta\vec{d}_4^*},g^{\sigma\vec{d}_5^*},g^{\sigma\vec{d}_6^*}\}$$

$\mathsf{KeyGen}(msk,(ID_1,\ldots,ID_j))\to sk_{ID}$: 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:

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

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:

$$\begin{aligned}&sk_{ID|ID_{j+1}} := \left\{g^{\gamma\vec{d}_1^*},g^{\xi\vec{d}_2^*},g^{\theta\vec{d}_3^*},g^{\theta\vec{d}_4^*},g^{\sigma\vec{d}_5^*},g^{\sigma\vec{d}_6^*}\right.\\ &K_1\cdot g^{y_1'\gamma\vec{d}_1^*+w_1'\xi\vec{d}_2^*+\omega_1^1ID_1\theta\vec{d}_3^*-\omega_1^1\theta\vec{d}_4^*+\omega_2^1ID_1\sigma\vec{d}_5^*-\omega_2^1\sigma\vec{d}_6^*}\\ &\ldots,K_j\cdot g^{y_j'\gamma\vec{d}_1^*+w_j'\xi\vec{d}_2^*+\omega_1^jID_1\theta\vec{d}_3^*-\omega_1^j\theta\vec{d}_4^*+\omega_2^jID_1\sigma\vec{d}_5^*-\omega_2^j\sigma\overrightarrow{d_6^*}}\\ &g^{y_{j+1}'\gamma\vec{d}_1^*+w_{j+1}'\xi\vec{d}_2^*+\omega_1^{j+1}ID_{j+1}\theta\vec{d}_3^*}\\ &\left.\cdot g^{-\omega_1^{j+1}\theta\vec{d}_4^*+\omega_2^{j+1}ID_{j+1}\sigma\vec{d}_5^*-\omega_2^{j+1}\sigma\vec{d}_6^*}\right\}\end{aligned}$$

$\mathsf{Encrypt}(pk,M,ID)\to\mathsf{ct}$: 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

$$C_0 := Me(g,g)^{\alpha_1s_1\vec{d}_1\cdot\vec{d}_1^*}e(g,g)^{\alpha_2s_2\vec{d}_2\cdot\overrightarrow{d_2^*}}$$

and

$$C_i := g^{s_1\vec{d}_1+s_2\vec{d}_2+t_1^i\vec{d}_3+ID_it_1^i\vec{d}_4+t_2^i\vec{d}_5+ID_it_2^i\vec{d}_6}$$

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:

$$M = C_0/B$$

G.1 Proof of Key-Indistinguishability

Theorem G.1. *The prime order variant of Lewko's scheme from [26] satisfies the key-indistinguishability property.*

The delegation algorithm (in Appendix G) in Lewko's prime order scheme [26] re-randomizes each exponent in a secret key. Each group element (the ones which are unique for an identity) in the key generated by KeyGen(), is of the form:

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

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 Delegate(), each group element (the ones which are unique for an identity) is of the form:

$K'_i = K^*_i \cdot g^{y'_i\gamma\vec{d^*_1}+w'_i\xi\vec{d^*_2}+\omega^i_1 ID_1\theta\vec{d^*_3}-\omega^i_1\theta\vec{d^*_4}+\omega^i_2 ID_1\sigma\vec{d^*_5}-\omega^i_2\sigma\vec{d^*_6}}$

Where K^*_i is the i'th group element of the key of the identity, which was used as an input for Delegate(). The following variables are sampled uniformly at random, $\omega^i_1, \omega^i_2 \xleftarrow{\$} \mathbb{Z}_p$ 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, $r^i_1 + \omega^i_1$ and $r^i_2 + \omega^i_2$ are uniformly random for each i. The keys generated via 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 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.

Theorem G.2. *The HIBE scheme from [13] satisfies the key-indistinguishability property.*

In a very similar manner to the key delegation algorithm in Lewko's prime order scheme, the delegation algorithm in the HIBE scheme from [13] also performs re-randomization of each exponent in a secret key. We give a brief description of the relevant algorithms below, and refer readers to [13] for a detailed description of the scheme:

- Setup $\left(1^\lambda, d\right)$ This algorithm takes in the security parameter λ, a depth parameter d and generates a bilinear pairing $\mathbb{G} := (q, G_1, G_2, G_T, g_1, g_2, e)$ for sufficiently large prime order q. The algorithm samples random dual orthonormal bases, $\left\{(\mathbb{D}_i, \mathbb{D}^*_i)\right\}_{i=0,\ldots,d} \leftarrow_{\mathrm{R}} \mathrm{Dual}\left(\mathbb{Z}^3_q, \mathbb{Z}^4_q, \ldots, \mathbb{Z}^4_q\right)$. It outputs the public parameters as
$\mathrm{PP} := \{ \{\mathbb{G}; g_T, g_1^{\mathbf{d}_{1,0}}, g_1^{\mathbf{d}_{3,0}}, \left\{g_1^{\mathbf{d}_{1,i}}, g_1^{\mathbf{d}_{2,i}}\right\}_{i=1,\ldots,d}, g_1^{\mathbf{d}^*_{1,0}}, \left\{g_1^{\mathbf{d}^*_{1,i}}, g_1^{\mathbf{d}^*_{2,i}}\right\}_{i=1,\ldots,d} \} \in G_T \times \left(G^3_1\right)^2 \times \left(G^4_1\right)^{2d} \times G^3_2 \times \left(G^4_2\right)^{2d}$
and the master key $\mathrm{MK} := g_1^{\mathbf{d}^*_{3,0}} \in G^3_2$.
- KeyGen $(\mathrm{PP}, \mathrm{MK}, (\mathrm{id}_1, \ldots, \mathrm{id}_\ell))$ This algorithm picks $r_1, \ldots, r_\ell, s_1, \ldots, s_\ell \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and sets $s_0 := s_1 + \cdots + s_\ell$. The secret key is computed as $\mathrm{SK}_{(\mathrm{id}_1,\ldots,\mathrm{id}_\ell)} := \{\mathrm{K}_0 := g_2^{-s_0\mathbf{d}^*_{1,0}+\mathbf{d}^*_{3,0}}, \left\{\mathrm{K}_i := g_2^{s_i\mathbf{d}^*_{1,i}+r_i\left(\mathrm{id}_i\mathbf{d}^*_{1,i}-\mathbf{d}^*_{2,i}\right)}\right\}_{i=1,\ldots,\ell} \} \in G^3_2 \times \left(G^4_2\right)^\ell$
- KeyDel $\left(\mathrm{PP}, \mathrm{SK}_{(\mathrm{id}_1,\ldots,\mathrm{id}_\ell)}, \left(\mathrm{id}'_1, \ldots, \mathrm{id}'_{\ell'}\right)\right)$ This algorithm picks $r'_1, \ldots, r'_{\ell'}, s'_1, \ldots, s'_{\ell'} \leftarrow_{\mathrm{R}} \mathbb{Z}_q$ and sets $s'_0 := s'_1 + \cdots + s'_{\ell'}$. The secret key is computed as $\mathrm{SK}_{\left(\mathrm{id}'_1,\ldots,\mathrm{id}'_{\ell'}\right)} := \{\mathrm{K}'_0 := \mathrm{K}_0 \cdot g_2^{-s'_0\mathbf{d}^*_{1,0}}, \left\{\mathrm{K}'_i := \mathrm{K}_i \cdot g_2^{s'_i\mathbf{d}^*_{1,i}+r'_i\left(\mathrm{id}'_i\mathbf{d}^*_{1,i}-\mathbf{d}^*_{2,i}\right)}\right\}_{i=1,\ldots,\ell'} \} \in G^3_2 \times \left(G^4_2\right)^{\ell'}$

Therefore, following the same argument as the proof of Theorem G.1, the distribution of the secret key for a particular identity generated via any sequence of delegations is the same as the distribution of a secret key for the same identity generated via KeyGen(). 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.

H ANALYSIS OF SIGNATURE SIZE FOR VARIOUS VALUES OF N

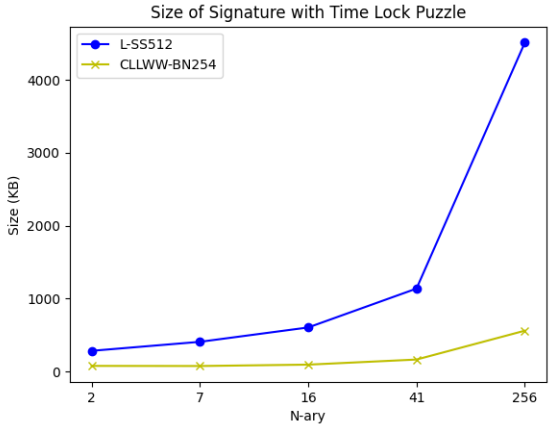

Figure 5: The signature size - including the encrypted functional key in the time lock puzzle - for a time-deniable signature scheme using the Lewko HIBE with SS512 and the CLLWW HIBE with BN254, varying values of N

I ON THE NECESSITY OF TIME-LOCK ENCRYPTION

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 [18, 21], we show that time-deniable signatures imply time-lock puzzles.

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

[8] When supplemented with a computational reference clock, it is known to imply time lock puzzles [30].

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 $\sigma \leftarrow \mathsf{DS.Sign}(sk, m, t)$. 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 $(x, w) \in R$. For a detailed discussion on witness encryption and extractable security we refer readers to the work [21] of Goldwasser et al. 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.

J ON THE NECESSITY OF SECURE TIMESTAMPS

Recall that in our definition, the 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 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 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 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 Sign/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.

K EPOCHAL SIGNATURES

We recall the unforgeability game and definitions from the work [24] 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 $O_{\mathsf{Evolve}}(\cdot)$ 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 $O_{\mathsf{Evolve}}(sk, \cdot)$ query, the adversary also gets access to the corresponding sign oracle for epoch e, $O_{\mathsf{Sign}}(sk_e, \cdot)$.
(2) The sign oracle $O_{\mathsf{Sign}}(sk_e, \cdot, \cdot)$ 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 $\mathbf{Adv}_{\mathcal{A}}(1^\lambda, \Delta t) = \Pr\left[\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}(1^\lambda, \Delta t, E, V) = 1\right]$.

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

Definition K.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 $\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}$:

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

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

K.1 Faulty Epochal Signature Construction

Given a secure epochal signature construction Σ, we use it to construct another secure epochal signature scheme Σ' which has undesirable properties as discussed in Section 3. However, due to certain properties implicit in the definition of an epochal signature scheme, the scheme we present as the counter example is fairly intricate. We begin with an informal description of the counter example, which suffices for a relaxed notion of epochal signatures. We build upon this to present our final scheme Σ' - we formally argue that Σ' (i) is a secure epochal signature scheme; and (ii) is not a time-deniable signature scheme.

We first consider a counter example that satisfies a weaker but still reasonable notion of deniability where the judge never gets access to any secret key material. In this setting, our counter example is fairly simple: each epoch e has a special trigger message m^*_e associated with it. If a signature ever needs to be made on m^*_e, in epoch e, then the signature contains the master secret key msk of the scheme. Included with every signature is a time lock puzzle that holds the special trigger message m^*_e, where the difficulty parameter on the puzzle is slightly more than Δt. It is straightforward to see that this scheme is still secure under the ES unforgeability game: the epoch e will always expire by the time the adversary could attempt to use m^*_e by solving the time lock puzzle. However, in the unforgeability game of time-deniable signatures, such a scheme is trivially defeated by an adversary in the pre-processing phase since the time lock puzzle can be solved during this phase.

The main problem with this counter-example is that the construction does not satisfy *perfect deniability*. Perfect deniability requires that one can simulate a *single* signature perfectly without revealing whether or not the signature was a forgery. Specifically in our counter example, we must ensure that one reply can never give away msk. To accomplish this, we encrypt msk under a key that is (2,2) secret shared. In order to recover the key, the adversary must make *two* queries which is explicitly disallowed in the deniability definition of epochal signatures. This ensures that only one share of the key is ever recovered, and thus we can simulate this share correctly without knowledge of msk. The final construction is presented in Figure 6. We would like to emphasize that this counter example is meant to show weaknesses in the unforgeability definition and that almost all of the complexity is added because of the deniability definition.

Theorem K.2. *The epochal signature construction in figure 6 is unforgeable under the security game of definition K.1*

Proof. We prove this by a standard hybrid argument starting with the real unforgeability game of $\mathcal{H}_0$. For any hybrid $\mathcal{H}_i$, the output of $\mathcal{H}_i$ is considered to be the challenger's output bit i.e. the adversary's success probability. This is related but not necessarily equal to $\mathbf{Adv}^{\mathcal{H}_i}_{\mathcal{A}}$. Therefore, the statement $\mathcal{H}_i \approx \mathcal{H}_{i+1}$ means that the probability of $\mathcal{A}$ succeeding between two different hybrid games is negligible. For security, we want $\mathbf{Adv}^{\mathcal{H}_0}_{A}(\lambda) \leq \text{negl}(\lambda)$ where $\text{negl}(\lambda)$ is a negligible function in the security parameter λ. Where details are omitted from the description of hybrid $\mathcal{H}_i$, it is assumed that they are the same as in $\mathcal{H}_{i-1}$.

$\mathcal{H}_1$: Guess the challenge epoch e^* of $\mathcal{A}$ by uniformly choosing an epoch in $\{1, \ldots, E\}$ where E is the maximum number of epochs. If the guess is incorrect, the challenger's output is 0.

Analysis:

$$\mathbf{Adv}^{\mathcal{H}_1}_{\mathcal{A}}(\lambda) = \frac{1}{E} \cdot \mathbf{Adv}^{\mathcal{H}_0}_{\mathcal{A}}(\lambda)$$

□

Let r_0, r_1, r_2, r_3 be the ephemeral randomness used in an epoch e. Consider the following hybrid,

$\mathcal{H}_3$: If $\exists e \in \{1, \ldots e^*\}$ such that $\mathcal{A}$ queries the sign oracle on messages r_0 or r_1, the challenger outputs 0.

Let F be the event that on at least one epoch $e \in \{1, \ldots e^*\}$, $\mathcal{A}$ queries on one or both of r_0 and r_1. If $\Pr[F] \leq \text{negl}(\lambda)$, then $\mathcal{H}_1 \approx \mathcal{H}_3$.

Lemma K.3. $\Pr[F] \leq \text{negl}(\lambda)$

We first provide a sequence of hybrids $\mathcal{H}^2_{i,j,k}$ where $i \in \{1, \ldots, e^*\}$, $j \in \mathbb{N}, k \in \{0, 1\}$.

$\mathcal{H}^2_{i,j,k}$: Let r_0, r_1, r_2, r_3 be the ephemeral randomness used in epoch i. If j is not 0, on the j^{th} sign query of $\mathcal{A}$ the time lock puzzle that normally holds r_k instead encrypts another random value $\hat{r_k}$ where $\hat{r_k} \neq r_k$.

Note that $\mathcal{H}_1 = \mathcal{H}^2_{1,0,k}$ for any k. For a given epoch i, we will be concerned with the possibly infinite sequence of hybrids $(\mathcal{H}^2_{i,0,0}, \mathcal{H}^2_{i,0,1}, \mathcal{H}^2_{i,1,0}, \mathcal{H}^2_{i,1,1}, \ldots)$ If this sequence is actually *finite* with the last query of $\mathcal{A}$ in i being $r(\lambda)$, then $\mathcal{H}^2_{i,r(\lambda),1} = \mathcal{H}^2_{i+1,0,0}$.

Claim 6. $\forall i \in \{1, \ldots e^*\}, \forall j \in \{1, \ldots, \}, \mathcal{H}^2_{i,j-1,1} \approx \mathcal{H}^2_{i,j,0}$

Suppose this is not true and there exists a distinguisher D with non-negligible advantage that distinguishes between $\mathcal{A}$'s success probability in $\mathcal{H}^2_{i,j-1,1}$ and $\mathcal{H}^2_{i,j,0}$ that outputs 0 when it thinks $\mathcal{A}$'s success is from $\mathcal{H}_{i,j-1,1}$ and 1 otherwise. Consider the following TimeLock adversary $\mathcal{B}$ which uses $\mathcal{A}$ and D to break the security of the TimeLock.

Description of TimeLock adversary $\mathcal{B}$:

- Execute $\Sigma.\text{KeyGen}(1^{\lambda}, \Delta t, E, V)$ to get pk, sk. Give pk to $\mathcal{A}$
- On queries in epochs $e > i$, respond as normal.
- For the j^{th} query in i, sample m uniformly from $\mathcal{R}$ subject to the constraint that m is not equal to r_0. Send to the TimeLock challenger (r_0, m). Receive s. Construct $\sigma, c'_{tlp}, c_{sk}, r'$ as normal and send to $\mathcal{A}$ the signature $\sigma, s, c'_{tlp}, c_{sk}, r'$.
- Let b' be the output of $\mathcal{A}$. Send $b = b'$ to D. If D outputs $\hat{b}$ respond with $\hat{b}$.

Analysis:

The probability that $\mathcal{B}$ wins is equivalent to the probability of D distinguishing correctly. Since D by assumption has non-negligible advantage so does $\mathcal{B}$, in contradiction to the security of

KeyGen$(1^\lambda, \Delta t, E, V)$:
$pk, sk \leftarrow$
Σ.KeyGen$(1^\lambda, \Delta t, E, V)$
return (pk, sk)

Evolve(sk'_e) :
if $e == 0$:
  $sk_e \leftarrow \mathsf{sk}'_e$
  return Σ.Evolve(sk_e)
$\begin{pmatrix} sk_e, r_0, \\ r_1, r_2, r_3 \end{pmatrix} \leftarrow$ parse(sk'_e)
$\begin{pmatrix} pinfo_{e+1}, \\ sk_{e+1} \end{pmatrix} \leftarrow \Sigma$.Evolve$(sk_e)$
$r_0, r_1, r_2, r_3 \leftarrow \mathcal{R}$
return $pinfo_{e+1}, (sk_{e+1}, r_0, r_1, r_2, r_3)$

Verify(pk, e, σ', m) :
if $e == 0$:
  $\sigma \leftarrow \sigma'$
else:
  $\sigma, \_, \_, \_, \_ \leftarrow \sigma'$
return Σ.Verify(pk, e, σ, m)

Sign(sk'_e, m) :
if $e == 0$:
  $sk_e \leftarrow$ parse(sk'_e)
  return Σ.Sign(sk_e, m)
$\begin{pmatrix} sk_e, r_0, \\ r_1, r_2, r_3 \end{pmatrix} \leftarrow$ parse(sk'_e)
$\sigma \leftarrow \Sigma$.Sign(sk_e, m)
$c_{sk} \leftarrow sk_e \oplus (r_2 \oplus r_3)$
$c_{tlp} \leftarrow$ TimeLock.Gen$(1^\lambda, \Delta t + \epsilon, r_0)$
$c'_{tlp} \leftarrow$ TimeLock.Gen$(1^\lambda, \Delta t + \epsilon, r_1)$
if $m == r_0$:
  $r' \leftarrow r_2$
else if $m == r_1$:
  $r' \leftarrow r_3$
else:
  $r' \leftarrow \mathcal{R}$
return $\sigma, c_{tlp}, c'_{tlp}, c_{sk}, r'$

Figure 6: Our counterexample ES construction Σ' built from another secure epochal signature scheme Σ. The domain of $\mathcal{R}$ is $\{0,1\}^\lambda$ and $\forall i \in \{0,1,2,3\}, r_i \in \mathcal{R}$. ϵ can be any value that is not 0, but we are especially interested in *small* ϵ.

the TimeLock.

Claim 7. $\forall i \in \{1, \ldots e^*\}, \forall j \in \mathbb{N}, \mathcal{H}^2_{i,j,0} \approx \mathcal{H}^2_{i,j,1}$

Proof sketch. The argument here is equivalent to the previous one, except instead of $\mathcal{B}$ challenging on (r_0, m) the challenge is (r_1, m). □

In order to be a successful adversary, $\mathcal{A}$ must run in polynomial time, which means that the number of queries that $\mathcal{A}$ can make in any particular epoch is also bound by some polynomial. Therefore, $\forall i$ the sequences $(\mathcal{H}^2_{i,0,0}, \mathcal{H}^2_{i,0,1}, \ldots)$ must also be bounded by poly(λ). Since $e^* \leq E \in$ poly(λ) is also polynomially bounded by the security parameter and the last hybrid in sequence $i-1$ is actually the first hybrid in i, there is a negligible difference between $\mathcal{H}^2_{1,0,0}$ and the last hybrid in the sequence beginning with $\mathcal{H}^2_{e^*,0,1}$.

The probability that $\mathcal{A}$ asks for either appropriate trigger in any of the epochs $e = 1 \ldots e^*$ can now be calculated by multiple union bounds. For an epoch i the probability that $\mathcal{A}$ guesses either r_0 or r_1 is $\frac{2}{|\mathcal{M}|}$ where $\mathcal{M}$ is the space of all possible messages that can be signed. If $\mathcal{M} = \{0,1\}^\lambda$ the probability this happens in any epoch before or at the challenge epoch is upper bounded by $\frac{2 \cdot e^*}{2^\lambda}$ which is negligible. Let $e^* = r(\lambda)$ and $q(\lambda)$ be the maximum number of queries $\mathcal{A}$ asks for in any epoch. Then we have

$$\begin{aligned} \Pr[F] &= q(\lambda) \cdot e^* \cdot \mathbf{Adv}^{\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{TL}}}_{\mathcal{B}}(\lambda) + \frac{2e^*}{2^\lambda} \\ &= q(\lambda) r(\lambda) \mathbf{Adv}^{\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{TL}}}_{\mathcal{B}}(\lambda) + \frac{2r(\lambda)}{2^\lambda} \end{aligned}$$

By assumption $\mathbf{Adv}^{\mathbf{Exp}^{\mathsf{IND}}_{\mathsf{TL}}}_{\mathcal{B}}(\lambda)$ is negligible and $\frac{1}{2^\lambda}$ is clearly negligible therefore $\Pr[F]$ is as well. This completes our proof of the lemma.

Recall that in $\mathcal{H}_3$ we know that $\mathcal{A}$ will not ask for a query on the trigger message r^* during the challenge epoch e^* or any earlier epoch. We can now argue security by reducing to the security of the original epochal signature scheme ES′. Let $\mathcal{B}$ be an adversary for the ES unforgeability game that is constructed from $\mathcal{A}$ in $\mathcal{H}_3$ as described below.

Description of ES adversary $\mathcal{B}$:

- Receive pk from the challenger and pass pk on to $\mathcal{A}$
- For any evolve query asked by $\mathcal{A}$ before or during e^*, make the same query to the challenger. Receive $pinfo_e$. Re-sample new randomness r_0, r_1, r_2, r_3. Send $pinfo_e$ to $\mathcal{A}$
- For any sign query m for epoch $e \leq e^*$, make the same query to the challenger and receive σ. Forward σ to $\mathcal{A}$ along with correctly strutured values $c_{tlp}, c'_{tlp}, c_{sk}, r'$
- When $\mathcal{A}$ gives forgery (m^*, σ^*), parse out the first component $\bar{\sigma}$ and give the forgery $(m^*, \bar{\sigma})$ to the challenger

Analysis: It should be clear that in this case if $\mathcal{A}$ succeeds $\mathcal{B}$ must as well since $\mathcal{A}$ never asks for a query on either r_0 or r_1 in the relevant epochs before the forgery and $\mathcal{B}$ is merely making the same queries to its challenger that $\mathcal{A}$ is. Therefore,

$$\mathbf{Adv}^{\mathcal{H}_3}_{\mathcal{A}}(\lambda) = \mathbf{Adv}^{\mathbf{Exp}^{\mathsf{UNF}}_{\mathsf{ES}}}_{\mathcal{B}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Since the advantage of $\mathcal{A}$ in $\mathcal{H}_3$ is negligible, the construction is unforgeable.

Theorem K.4. *The epochal signature construction in figure 6 is deniable according to the definition of [24].*

We first give a description of the simulator Sim for the deniability game. It makes use of Σ.Sim which is the simulator for the deniability of Σ.

$\mathsf{Sim}(m, e_0, pinfo_{e_0+e_1})$:
- $r \leftarrow \mathcal{R}$
- $c_{tlp} \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t + \epsilon, r)$
- $c_{sk} \leftarrow \mathcal{R}$
- $r' \leftarrow \mathcal{R}$
- $\sigma \leftarrow \Sigma.\mathsf{Sim}(m, e_0, pinfo_{e_0+e_1})$
- if $e_0 == 0$ or $e_0 == 1$:
  - return σ
- return $\sigma, c_{tlp}, c_{sk}, r'$

In order to prove that our scheme is deniable we need the distributions of $\sigma, c_{tlp}, c_{sk}, r'$ to be indistiguishable when generated via Sign or Sim when the distinguisher has access to $pk, sk_{e_0+e_1}, pinfo_{e_0+e_1}$.

We can safely ignore σ because σ is conditionally independent from the tuple (c_{tlp}, c_{sk}, r') given sk and it does not reveal information on what it was derived from because of the deniability of Σ. We can also restrict our analysis to $e > 1$, since when $e = 1$, σ is the only component of the signature.

This can be simplified to us needing the following to hold for all messages m, for all valid epochs $e_0 \neq 0$ and $e_0 \neq 1$, for all valid e_1, $\forall V$, and all key generation outputs $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta t, E, V)$:

$$\left\{\begin{array}{l} r_0, r_1, r_2, r_3 \leftarrow \mathcal{R}, \\ (\_, sk_{e_0}) \leftarrow \mathsf{Evolve}^{e_0}(sk), \\ (pi_{e_0+e_1}, sk_{e_0+e_1}) \leftarrow \mathsf{Evolve}^{e_1}(sk_{e_0}), \\ c_{tlp} \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t + \epsilon, r_0), \\ c'_{tlp} \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t + \epsilon, r_1) \\ c_{sk} \leftarrow (r_2 \oplus r_3) \oplus sk_{e_0}, r' \leftarrow \mathcal{R}^{***} \end{array} : \begin{array}{l} c_{tlp}, c'_{tlp}, c_{sk}, r', \\ pk, sk_{e_0+e_1}, pi_{e_0+e_1} \end{array}\right\} \approx$$

$$\left\{\begin{array}{l} r_0, r_1 \leftarrow \mathcal{R}, \\ (pi_{e_0+e_1}, sk_{e_0+e_1}) \leftarrow \mathsf{Evolve}^{e_0+e_1}(sk) \\ c_{tlp} \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t + \epsilon, r_0), \\ c'_{tlp} \leftarrow \mathsf{TimeLock.Gen}(1^\lambda, \Delta t + \epsilon, r_1) \\ c_{sk} \leftarrow \mathcal{R}, r' \leftarrow \mathcal{R} \end{array} : \begin{array}{l} c_{tlp}, c'_{tlp}, c_{sk}, r', \\ pk, sk_{e_0+e_1}, pi_{e_0+e_1} \end{array}\right\}$$

The asterisks on the left distribution for r' denote that the value of r' is mostly random, except when $m = r_0$ or $m = r_1$. In that case, $r' = r_2$ or $r' = r_3$ respectively.

Since $V \geq 1$, the updated key $sk_{e_0+e_1}$ does not contain the randomness used in epoch e_0. However, both "trigger" messages, are made available to the judge by breaking the time lock puzzles c_{tlp} and c'_{tlp}. Luckily this, at most, gives the judge access to one of r_2 or r_3: the judge only gets to see the output of one signing query so if the judge uses the message $m = r_0$ or $m = r_1$ it receives either r_2 or r_3 as r'. Recall that $c_{sk} = (r_2 \oplus r_3) \oplus sk_{e_0}$ whenever Sign is used. Because the judge can only get access to at most one share, c_{sk} is indistinguishable from a random element of $\mathcal{R}$. This also means that the random pad r' is independent of c_{sk} to the perspective of the judge, regardless of if it is supposed to be a share of the one time pad or not. Therefore the joint distribution of (c_{sk}, c_{tlp}, r') in the left-hand side of the equation is one where each value is independent from the others, c_{sk} and r' are uniform, and c_{tlp} locks a uniformly random value. This is precisely what the right hand side is and thus completes our proof.

K.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 7.

Theorem K.5. *The* ES *scheme presented in Figure 7 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 $\mathcal{A}$:

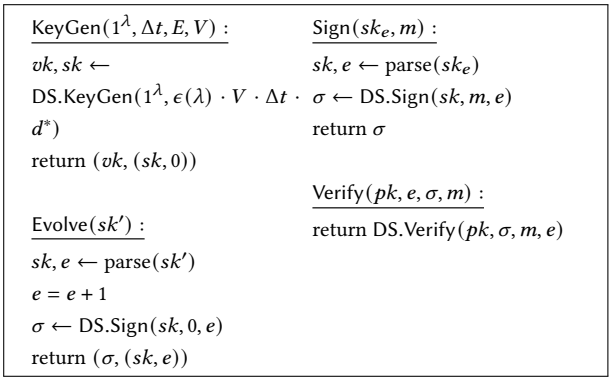

Figure 7: An ES construction from a time-deniable signature scheme DS. $\epsilon(\lambda)$ is the admissibility parameter for the time-deniable signature scheme. The sentinel message for $pinfo$ is $m = 0$.

- Receive vk from the challenger. Give vk to $\mathcal{B}$. Initialize a counter $ctr = 0$ and the wall clock time $t_{init} = \text{now}()$.
- On queries O_{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 O_{Sign} on message 0 to receive σ. Return σ as $pinfo$. Else do not advance the counter and output $\perp$.
- On queries O_{Sign} from $\mathcal{B}$ with message m, query the challenger's O_{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, size($\mathcal{B}$) $\in$ poly(λ) and since $\mathcal{A}$ is also doing poly(λ) work while interacting with $\mathcal{B}$, we have that size($\mathcal{A}_0$) $\in$ poly(λ). 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. 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 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 K.6. *The* ES *scheme presented in Figure 7 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 O_{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 ES.Sign in Figure 7 which is simply DS.Sign. When $b = 1$, the output is produced by the simulator $\mathcal{S}$ in the ES deniability game which is the equivalent to DS.AltSign in our construction. Therefore, the distributions $\mathcal{J}$ sees are correct and if $\mathcal{J}$ is successful in distinguishing then so is $\mathcal{A}$.