# KeyForge: Non-Attributable Email from Forward-Forgeable Signatures

Michael A. Specter, *MIT;* Sunoo Park, *MIT & Harvard;*
Matthew Green, *Johns Hopkins University*

https://www.usenix.org/conference/usenixsecurity21/presentation/specter-keyforge

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

# KeyForge: Non-Attributable Email from Forward-Forgeable Signatures

Michael A. Specter
*MIT*

Sunoo Park
*MIT & Harvard*

Matthew Green
*Johns Hopkins University*

## Abstract

Email breaches are commonplace, and they expose a wealth of personal, business, and political data whose release may have devastating consequences. Such damage is compounded by email's strong attributability: today, any attacker who gains access to your email can easily prove to others that the stolen messages are authentic, a property arising from a necessary anti-spam/anti-spoofing protocol called DKIM. This greatly increases attackers' capacity to do harm by selling the stolen information to third parties, blackmail, or publicly releasing intimate or sensitive messages — all with built-in cryptographic proof of authenticity.

This paper introduces *non-attributable email*, which guarantees that a wide class of adversaries are unable to convince discerning third parties of the authenticity of stolen emails. We formally define non-attributability, and present two system proposals — KeyForge and TimeForge — that provably achieve non-attributability while maintaining the important spam/spoofing protections currently provided by DKIM. Finally, we implement both and evaluate their speed and bandwidth performance overhead. We demonstrate the practicality of KeyForge, which achieves reasonable verification overhead while signing faster and requiring 42% *less* bandwidth per message than DKIM's RSA-2048.

## 1 Introduction

Email has long been the world's largest messaging scheme, used ubiquitously for personal, industry, and government communication. As such, it is a valuable target for attack: a user's account is a trove of sensitive information, unauthorized access to which enables spam, fraud, blackmail, and other abuse.

To help protect users from spam and fraud, the IETF developed a widely-adopted standard called DomainKeys Identified Mail (DKIM) [16]. DKIM's goal is to assure the receiving server that each incoming message was really sent from the domain it appears to be from, enabling inter-domain accountability in case of spam and easy detection of spoofed messages. DKIM's protocol is simple: the originating server cryptographically signs each outgoing email's contents and metadata, allowing the receiving server to verify the message after looking up the sending server's public key via DNS.

While DKIM was an important innovation that continues to be critical to the email ecosystem, its design came with an unintended side-effect: namely, email thieves can credibly convince any third party that stolen messages are authentic and unmodified via DKIM signatures from a reputable service provider. This increases incentives to break into email accounts, as a successful attacker can credibly (and anonymously) sell, publish, or use the stolen data for blackmail.

Email attributability has had real-world impact. For example, Wikileaks publicly asserts [56] that it relies on DKIM signatures to confirm the veracity of their publications: Wikileaks leveraged DKIM to authenticate messages stolen from the Democratic National Committee (DNC) and Hillary Clinton's campaign chairman during the 2016 U.S. presidential election season [55]. Because of DKIM, any third party could easily confirm the legitimacy of these stolen messages using public keys tied to Google and Microsoft's email services, despite the information's questionable origin. Indeed, the practice of using DKIM to verify unauthorized email leaks has now become a standard journalistic practice [40, 46], with the Associated Press releasing a software tool for this purpose [6].

DKIM's attributability problem has been recognized but unsolved for some time. Jon Callas, one of the original authors of the DKIM RFC, has publicly stated that attributability is an unintended design flaw of the protocol [17, 18], and has since suggested a number of ways to mitigate its impact, but notes that proposals at the time of his writing were insufficient or impractical [19]. Other researchers also flagged the issue as early as 2004, e.g., Adida *et al.* [2], Unger *et al.* [52], and Bellovin [9]; however, designing a practical, non-attributable DKIM replacement has remained an open question.

It is alarming that an unintended result of an ubiquitous messaging protocol has produced a *scalable, by-default* system for credible propagation of illicitly obtained private messages. The specific DNC incident might well have happened with or without DKIM: for a high-value target, interested parties would likely seek to verify the stolen emails in various ways, including non-technical methods (e.g., journalistic corroboration, cross-checking timestamps, geolocation, etc). But just the possibility of manual verification — a possibility that has existed since handwritten letters — is a stark contrast from the easy, inbuilt attribution that has *unintentionally* become ingrained in today's email ecosystem.

Public figures are not the only victims of email breaches; new reports of email theft seem to surface every few weeks. Astoundingly, all of Yahoo!'s 3 billion email accounts were compromised in a 2013 breach [49]. Although Yahoo!'s users have been spared public dissemination of their messages, others (e.g., Sony and Stratfor), have been less fortunate [53, 54]. Attackers appear to have diverse motives, ranging from financial gain — e.g., selling patient healthcare data gleaned from emails [33] — to industrial espionage and monitoring political dissidents and foreign officials [28].

In light of the potential harm to users, it would be irresponsible to let DKIM's unintended side-effect of attributability remain unscrutinized: if attributability is to remain a feature of DKIM, it should be as a result of a deliberate decision that takes into account the range of technically feasible alternatives. With the above as motivation, we ask:

> *Is it possible to mitigate the potential harms of attributability in DKIM while maintaining the system's efficient spam and spoofing resistance?*

An initial intuition may be that attributability of stolen email is an unavoidable side effect of spam and spoofing resistance, given the indirect and decentralized nature of email: it is intuitively unclear how a recipient with no communication to the sending server can be certain of a message's origin without also gaining the ability to convince a third party of the same. Under certain conditions, this intuition amounts to an impossibility. Yet, perhaps surprisingly, our work shows that modern cryptography can reconcile the apparently conflicting goals of spam protection and non-attributability. We construct efficient protocols that achieve the important security guarantees that DKIM provides, while simultaneously *guaranteeing non-attributability* of stolen email. Further, we show that configurations of our protocols are *practical* for deployment on the Internet today, achieving reasonable efficiency and bandwidth overhead.

## 1.1 Key Ideas

There are two main ideas underlying our proposals: *delayed universal forgeability* and *immediate recipient forgeability*.

**Delayed universal forgeability.** This approach ensures that signatures with respect to past emails "expire" after a time delay $\Delta$ and thereafter become forgeable by the general public (i.e., arbitrary outsiders or non-parties). This property ensures that no attribution will be credible after the time delay has elapsed. We call this property *delayed universal forgeability*. As long as $\Delta$ is set larger than the maximum viable time for email latency, the signature will still be convincing to the recipient at the time of receipt, thus maintaining the spam and spoofing-resistance of DKIM.

Signatures that possess delayed universal forgeability retain all the unforgeability properties of a standard signature scheme, until the set time $\Delta$ has passed. Thus in cases where an attacker gains access to email and shows it to a third party within $\Delta$ time after the email was sent, a third party will be convinced of the email's authenticity. Effectively, delayed universal forgeability protects against adversaries that compromise an email account by breaking in and taking a snapshot ("after-the-fact attacks"), but not adversaries that fully control an email account and monitor its email in real time ("real-time attacks"). After-the-fact attacks cover a broad range of realistic attacks, for example, including many data breaches. Next, we discuss how we address real-time attacks.

**Immediate recipient forgeability.** Suppose that the fact of access to a particular client account implies the ability to forge messages from arbitrary other servers *to that recipient only*: that is, the ability to obtain valid DKIM signatures on email content and metadata of one's choice. We call this *immediate recipient forgeability*. Importantly, the recipient constraint ensures the inability to impersonate any other server for the purposes of email addressed to *other* recipients, thus maintaining DKIM's spam and spoofing-resistance. This undermines the credibility of attackers claiming ongoing access to a particular email account and attempting to convince third parties of the authenticity of emails supposedly sent to (and from) that account — even for real-time attacks, which may publish allegedly-incoming emails immediately as they are received.

Recipient forgeability is weaker than universal forgeability in the following sense: published emails credibly reveal that the attacker has gained access to some users' key material, although not that the email content is authentic. Thus, recipient forgeability is not enough by itself; the two definitions are complementary and incomparable.

**Combining both ideas.** Our protocols attempt to achieve the "best of both worlds," by providing universal forgeability when possible, and falling back on immediate recipient forgeability when necessary. Section 3 defines our threat model, discusses its limitations, and formalizes immediate recipient forgeability and delayed universal forgeability.

## 1.2 Overview of Solutions

This paper constructs and evaluates two base protocols KeyForge and TimeForge, and two enhanced variants KeyForge$^+$ and TimeForge$^+$ (which consist of the respective base protocol with a modified signing algorithm and one additional sub-protocol). The two base schemes can be seen as two different approaches to building a new type of signature scheme that we introduce: *forward-forgeable signatures (FFS)*.

**Forward-forgeable signatures.** An FFS is a digital signature scheme equipped with a method to selectively disclose signature-invalidating "expiry information" for past signatures without similarly damaging the public key for future signatures. *Succinctness* of FFS is a measure of efficiency of disclosure. We present two constructions of FFS, which are the key building blocks of KeyForge and TimeForge respectively. FFS may be of independent interest as a signature primitive for other applications.

**KeyForge.** Our first proposal, KeyForge (§5.1), achieves *delayed universal forgeability* by publishing signing keys after a delay $\Delta$. KeyForge relies on an FFS based on hierarchical identity-based signatures (HIBS), which achieves logarithmic succinctness. As a result, KeyForge can efficiently distribute forging keys with minimal bandwidth.

**TimeForge.** Our second protocol, TimeForge (§5.2), assumes a *publicly verifiable timekeeper (PVTK)* model in which a trusted timekeeper periodically issues publicly verifiable timestamps. In a nutshell, the idea of TimeForge is to substitute each signature on a message $m$ at time $t$ with a succinct zero-knowledge proof of the statement $S(m) \vee T(t+\Delta)$, where: $S(m)$ denotes knowledge of a valid signature by the sender on $m$ and $T(t+\Delta)$ denotes knowledge of a valid timestamp for a time later than $t+\Delta$. Including $T(t+\Delta)$ ensures *delayed universal forgeability*. TimeForge can be described as a forward-forgeable signature scheme in the PVTK model.

**KeyForge$^+$/TimeForge$^+$.** The enhanced protocols (§5.3) consist of the respective base protocols with the following modifications: (1) an additional protocol, called *forge-on-request*, that allows parties to request forged emails addressed *only to the requester herself* under limited circumstances; and (2) for multiple-recipient emails, a new signature is produced for each recipient domain (unlike the base protocols and DKIM, which produce one signature per outgoing email).

Among our protocols, KeyForge is the most efficient and would necessitate the least change to existing infrastructure. KeyForge$^+$ and TimeForge$^+$ are alternative approaches showing the feasibility of addressing stronger threat models though at significant overhead (in fact, certain overhead is unavoidable in the stronger threat model; see §3). TimeForge could become more practical with advances in the fast-moving area of non-interactive proofs.

**Summary of our Contributions.**

1. We define non-attributability in store-and-forward email systems, and propose two *system designs* — KeyForge (§5.1), and TimeForge (§5.2) — that achieve this goal.
2. We *implement* KeyForge and TimeForge and evaluate their signing, verification, and bandwidth costs, and show that KeyForge has acceptable bandwidth and processing overhead for practical deployment (§6).
3. We provide formal *definitions* for email non-attributability and prove that our constructions realize them.
4. Of independent interest, we give *provably secure constructions* of a new cryptographic primitive, *succinct forward-forgeable signatures (FFS)*.

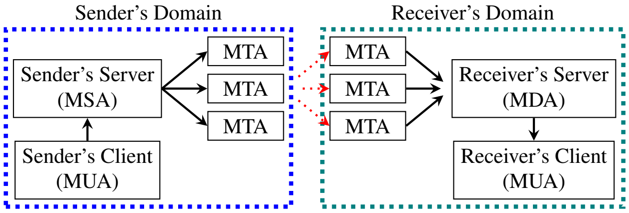

Figure 1: Simplified email routing infrastructure

## 2 Background on Email

This section introduces basic terminology of mail routing (as defined in RFC 5598 [24]) and describes how email infrastructure necessitates certain system requirements.

As described in Figure 1, email uses an asynchronous "store and forward" routing protocol built on top of TCP/IP. Users first establish a relationship with a trusted email service provider, called a Mail Submission Agent (MSA) on the sender side and a Mail Delivery Agent (MDA) on the receiver side. The user's email client is called a Mail User Agent (MUA). Email originates from an MUA, and arrives at the user's trusted MSA. Depending on the system's configuration, the MSA may send the message to intermediary Mail Transfer Agents (MTAs) it trusts. Eventually, as the message leaves the sending server's domain, an MTA will perform a DNS lookup to discover which MTAs are authorized to process messages for the receiving domain, and the email is then sent via SMTP to one of these destination MTAs. After a number of hops depending on the sending and receiving organizations' infrastructure, the email reaches the receiver's MDA, which is responsible for verifying the message for the receiver's MUA.

### 2.1 Email Authentication

The IETF has developed a number of standards that allow domains to sign and verify incoming and outgoing messages. Next, we overview the three that have seen appreciable adoption: DKIM, SPF, and DMARC. The IETF has also proposed an experimental protocol called ARC, which allows intermediaries to modify email messages in an authenticated way. We discuss the implications of ARC in the full version [48].

**DKIM.** DomainKeys Identified Mail (DKIM) is an IETF standard that requires an MSA to sign outgoing email, and an MDA to verify that email by looking up the MSA's public key in the DNS. This procedure is described informally below:

1. *Setup:* The MSA generates a key pair and uploads the public key to the DNS in a TXT record.

2. *Sign:* The MSA adds the location of its public key to the email's metadata (or *header*), as well as additional metadata needed for signature verification, then signs the email and headers with its private key.[1]
3. *Verify:* On receipt, the MDA does a DNS lookup for the MSA's public key, and uses it to verify the signature.

**SPF.** The Sender Policy Framework (SPF) ensures that intermediary MTAs are permitted to send and receive messages as a part of the domain. This solves a somewhat orthogonal problem to DKIM: SPF provides spoofing protection by limiting what IP addresses are valid accepting MTAs.

**DMARC.** An SPF or DKIM failure as a result of a misconfiguration is indistinguishable from a failure due to an attempted message spoofing, and neither DKIM nor SPF provide mechanisms for alerting the sending domain that there has been a problem. DMARC solves this by adding a DNS TXT record specifying to the receiver what it should do in the case of such failures (such as quarantine, reject, or accept the message despite the failure), as well as providing an email address to send aggregated statistics on such failures.

## 2.2 DKIM Replacement Constraints

This section overviews a number of demands on email that are not common to many other messaging systems. We find that these requirements make achieving email deniability and security uniquely difficult, and necessitate the new approach we describe in this paper.

**Indirectness by store and forward.** Email routing is a *store and forward* protocol in which messages are delivered indirectly via multiple hops, and routes, as well as the actual destination addresses, are often not known in advance. To quote the SMTP RFC [37], "[i]t is sometimes difficult for an SMTP server to determine whether or not it is making final delivery since forwarding or other operations may occur after the message is accepted for delivery." Obvious examples of indirectness include *mail forwarding* (in which users configure their MDA to forward email received from an account on one domain to another), and *remailers* (such as mailing lists, that act as MUAs initially); however, there are other, less obvious, places in the ecosystem where this occurs.[2]

For example, many organizations leverage *third-party MTAs* that they do not own as an initial hop between the Internet and the organization's self-hosted MDA/MSA.[3] These MTAs often provide security benefits to the MDA, such as protection from spam, malicious attachments, or DDoS attacks. While these intermediaries are allowed to quarantine messages or provide flow control to the MDA, under DKIM, they cannot undetectably modify or spoof emails.

In summary, email's indirect, store-and-forward system results in the following constraints: (1) final-destination information (e.g., addresses, keys) may be unknown to the sender, and (2) an MDA may not be certain whether it is the final destination of a message.

**Throughput and scalability.** Email is an any-mesh ecosystem in which any domain owner must be able to set up the appropriate DNS records and interoperate with any other domain's servers. Further, larger domains may sign and verify hundreds to millions of emails per day, and throughput requirements often increase over time. Therefore, beyond good constants on signing and verification time, the service must scale: adding more resources should provide linear or better performance, and scalability in interconnection with other servers is crucial as well.[4]

Such scalability requirements indicate that certain types of overhead that would be trivial in other messaging contexts, (e.g., communication prior to sending a message or per-message round trips between servers), are unlikely to be viable for email. For example, it would be difficult to require the MDA to connect back to the original MSA for every email.

**Long-lived public keys.** One natural approach to short-lived signatures is to leverage correspondingly short-lived keys and publish each secret key at the end of its lifetime, or use short key sizes designed to be able to be brute-forced within the same period (see  [19]). This approach has been mentioned in passing outside of the context of email [12]. Unfortunately, too-frequent key rotation entails practical problems that render this tactic unworkable for DKIM. Rotating keys stored in DNS is an often manual process that introduces risk of misconfiguration that can cause stability issues, and storing large amounts of key material that must be published, maintained, and shared among several servers is organizationally difficult and increases risk of key theft. DNS results are also often cached, so replacing an individual record is slow and can yield inconsistent results. Finally, it is hard to bound the time for short keys to be broken by all threat actors.

**Incremental deployment.** Given the myriad existing email servers and the need for interoperability, we consider the majority of the email ecosystem to be entrenched. It would be difficult to require substantial changes to mail routing, and it is unrealistic that every actor would promptly switch to a new scheme. Instead, it is far more realistic that DKIM could be replaced by incrementally updating the signing algorithms.

[1] This usually includes a hash of the whole message, but the specification does allow for portions of the message to go unsigned. This is not default behavior for most DKIM applications, and has seen limited use in practice.

[2] Similarly, Mail Retrieval Agents (MRAs) like Getmail [22] behave like MUAs to an MDA, but may forward emails on to an alternate, final MDA. Popular email services like Gmail provide services that download messages from other domains via IMAP.

[3] Third-party MTAs are commonplace. We did an informal survey by scraping DNS MX records for the Alexa top 150k. Surprisingly few, 31,615, have an MX record, and 10,260 use an obvious third-party hosting service (e.g., Google's MTAs), leaving 21,615 that potentially self-host. Of the last category, 31.4% (6,793) are using a confirmed multi-hop third-party MTA. Raw results are in our repo in results.csv [1]. This is likely a conservative estimate, as few servers appear to have matching domain names.

[4] The IETF standard for DMARC [38] states that pre-sending agreements is a poor scalability choice for this reason. See also [50].

### 2.2.1 Resulting System Requirements

The particular constraints of email, described earlier in §2.2, rule out many natural approaches to non-attributability, including solutions that might be more feasible in other messaging environments. Since we treat email's indirect, store-and-forward nature as an entrenched property of the infrastructure, realistic proposals for email protocol modifications must not rely on sender use of final-destination information, such as addresses or keys ("Requirement 1" or "R1"). Moreover, due to the store-and-forward and scalability requirements, email protocols should avoid interactive sender-receiver (MDA–MSA) communication whenever possible; in particular, we consider roundtrip sender-receiver communication per email to be inviable ("R2"). Additionally, email protocols must have long-lived public keys ("R3").

Notably, *none* of the following approaches adhere to both the above requirements: interactive zero-knowledge proofs (violate R2); ring signatures (proposed for email non-attributability in [3, 12]) (violate R1); designated-verifier signatures (violate R1); short-lived keys with publication of secret keys after use (violate R3); and — importantly — systems based on deniable authenticated key exchange (DAKE) (which violates R2), such as OTR or Signal [12, 51, 52]. Indeed, both the OTR paper [12, §6] and a recent DAKE paper [51, §6.6] dedicate a full subsection to discussing the heightened challenges of non-attributability for email as compared to other messaging environments, and note that their proposals are not adequate for email due to its asynchronous, non-interactive, store-and-forward nature.

Finally, we note that the simple approach of relying on MDAs to delete DKIM header information after receipt is flawed *not only* because it fails to address our threat models (§3), which require security against *malicious or compromised recipients*, but also because it violates Requirement 1: relying on MDAs for deletion is untenable given that MDAs may not know if they are the final endpoint (and if not, the signatures must be kept for later verification).

**Summary.** A viable non-attributable replacement for DKIM must have: (1) compatibility with indirect, store-and-forward communication (in particular, no reliance on sender knowledge of final destination addresses or keys); (2) no requirement of sender-receiver interaction per email; (3) long-lived public keys; (4) no required behavior for MDAs that depends on whether they are the final destination; (5) little impact on other parts of the email ecosystem; and (6) good systems properties allowing for incremental, scalable deployment.

## 3 Model and Security Definitions

**Notation.** "PPT" means "probabilistic polynomial time." $|S|$ denotes the size of a set $S$. $[n]$ denotes the set $\{1,\dots,n\}$ of positive integers up to $n$, and $\mathbb{P}(\cdot)$ denotes powerset. $\approx_c$ denotes computational indistinguishability. $\tau||e$ denotes the result of appending an additional element $e$ to a tuple $\tau$.

- **Time** We model time in discrete time-steps and assume fairly consistent (say, within 3 mins) local clocks. This is realistic given NTP [15].
- **Synchrony** $\hat{\Delta}$ is an upper bound on the time required for email delivery. Our parameter settings depend on $\hat{\Delta}$, and our evaluation sets $\hat{\Delta}$ at 15 minutes (see § 5.1).
- **DNS** Our model assumes all parties and algorithms have access to DNS and can update their own DNS records.
- **Bulletin board** We assume each party has a way to publish persistent, updatable information retrievable by all other parties and algorithms. This could be via DNS or another medium, such as posting on a website. (Formally, this can be modeled as a global service $\mathsf{BB}$ that: (1) is initialized with an empty table of key-value pairs; (2) upon receiving a message in $\{\mathsf{write},\mathsf{append}\} \times \{0,1\}^*$ authenticated with respect to a public key $pk$, respectively (over)writes or appends $x$ to the value (if any) associated with key $pk$; and (3) upon receiving a message of the form $(\mathsf{lookup}, pk)$, responds with the value $x$ associated with $pk$ in the table, if any.)
- **Publicly verifiable timekeeping service (PVTK)** A PVTK is a global service, initialized with respect to public parameters $pp$, which maintains a monotonically increasing clock. At any time $t$, any party can query the PVTK to obtain a publicly verifiable (w.r.t. $pp$) proof $\pi_t$ that the current PVTK clock time is at least $t$, but such proofs are computationally hard to forge for future times $t' > t$.

In the context of the KeyForge family of protocols, all algorithms are assumed capable of interacting with $\mathsf{BB}$. In the context of the TimeForge protocols, all algorithms are instead assumed to be able to query a global PVTK. (To simplify notation, we do not write $A^{\mathsf{BB}}$ or $A^{\mathsf{PVTK}}$ explicitly; but these assumptions will be recalled in the respective sections.)

**Threat models.** We are concerned with attacks that disclose private communications obtained at the MDA (whether because the MDA is compromised or because it is malicious).

We consider two threat models, defined below. KeyForge and TimeForge achieve security against Threat Model 1, which targets scenarios where attackers may gain access to an email server but are unlikely to maintain access for extended periods. The enhanced protocols KeyForge$^+$ and TimeForge$^+$ achieve security against Threat Model 2, the stronger of the two threat models, which is necessary in settings where attackers' access may likely remain undetected for extended periods (e.g., advanced persistent threats).

**Threat Model 1.** (After-the-fact attacks) *Recipient honest at the time of email receipt, but is later compromised by an attacker that takes a snapshot of all stored email content.*

**Threat Model 2.** (Real-time attacks) *Recipient may be malicious at the time of email receipt, with ongoing and immediate intent to disclose received email content to third parties.*

**Ruling out trivial solutions.** A trivial and uninteresting way to achieve non-attributability, in either threat model, is not to sign emails at all. Of course, this is undesirable as it would undermine the spam- and spoofing-resistance for which DKIM was designed. Providing these guarantees is an implicit requirement throughout this paper. Moreover, since our threat models consider malicious receiving servers, any non-attributability that relies on receiving-server behavior — such as DKIM header deletion upon receipt — is unsatisfactory.

**Preventing real-time attacks requires interaction.** Any store-and-forward email protocol that both (1) allows recipients to verify the sending domain's identity and (2) is secure against *real-time* attacks (Threat Model 2) must be interactive, as more formally detailed in the full version [48]. Informally, in the store-and-forward model, a non-interactive protocol transcript (consisting of a single message from the sender), cannot depend on final-destination recipient information, so any operations (such as verification or forgery) that the verifier can run must also be executable by others. This also relates to the intuitive idea that someone who receives a single message $m$ convincing them of the message's origin must also be able to use $m$ to convince others of the same.

In contrast, security against *after-the-fact* attacks (Threat Model 1) is possible non-interactively, as KeyForge and TimeForge exemplify. KeyForge$^+$ and TimeForge$^+$ augment KeyForge and TimeForge with an interactive (two-message) protocol, which adds significant overhead and complexity to the non-interactive base protocols. The overhead of our constructions is furthermore minimal in certain respects: just two rounds of interaction, and the protocols do not *require* interaction on email receipt, but rather, introduce the *possibility* of interaction by an additional protocol (details in §5.3).

**What's outside our threat models?** While Threat Model 2 considers powerful real-time adversaries, it too has limits. Definitionally, and unsurprisingly, no deniability is possible against a global passive adversary that can be sure of observing all traffic as it flows over the network. As already mentioned, our threat models are not designed to provide non-attributability against adversaries directly observing email traffic, but rather against those to whom the adversaries might try to pass the stolen emails on.

Our threat models focus on attacks at the receiving server (MDA), because we believe this covers a wide, though not exhaustive, range of attack scenarios of interest. This notably excludes *malicious intermediaries (MTAs)*. Even though our threat models do not focus on MTA-based attacks, our protocols KeyForge$^+$ and TimeForge$^+$ do provide a partial non-attributability guarantee against malicious intermediaries (as discussed in §3.1). Nonetheless, malicious intermediaries pose a legitimate concern not fully addressed by this work; achieving stronger non-attributability guarantees against MTAs could be interesting future research.[5]

Finally, we note that our definitions do not necessarily provide non-attributability against adversaries that can preconfigure the receiving server with custom secure hardware (see also §3.1). We consider such attacks outside our threat model: i.e., we assume servers are compromised after physical setup.

We conclude this section with additional context and explanation for our modeling choices.

**Client-server trust.** Email clients rely heavily on their email servers. A malicious email server could easily and undetectably misbehave in many essential functions: e.g., drop incoming emails, modify outgoing emails (since typically, emails are not signed client-side), or falsify content and metadata of incoming emails (since typically, clients do not perform DKIM verification themselves). Since client-server trust is very high in practice, this paper treats the client and server as a single entity, and relatedly, our threat models do not consider malicious behavior by MSAs that aims to undermine non-attributability of their own clients' emails. (One might also argue such malicious behavior would quickly lose an MSA its clients.)

**Evidence-based credibility.** In a system where credibility is based on reputation rather than evidence — that is, where certain parties' statements are taken on faith, or believed simply because of who they are even without supporting evidence — a "reputable" party with the ability to eavesdrop on the communication channel would be able to undermine non-attributability by keeping traffic logs. Our model assumes mutually distrustful parties: i.e., that no party is taken simply on its word as just described. In other words, credibility in our model is evidence-based and not reputation-based.

**Systemic attributability vs. attributability by choice.** The goal of non-attributability is to empower users to choose whether or not their messages are attributable, to disincentivize email theft and misuse in contrast to attributability-by-default (see §1). We are not concerned with preventing attributability when correspondents desire it: e.g., for business transactions or contracts, correspondents may intentionally sign messages to ensure they are binding. Attribution by journalistic investigation is also outside our threat model: confirmation of selected documents by careful investigation is possible even with handwritten letters, but the current *systemic* attributability facilitates scalable, malicious attribution far beyond the handful of high-profile messages that might be published after arduous manual verification.

[5] It is also unclear how effective local MTA-based attacks would be to compromise entire email accounts; such attacks' effectiveness would likely depend on email routing configurations at the servers involved. By entire-account compromise we mean learning all stored emails and/or all real-time emails for a single account over an extended period, as opposed to learning only occasional emails from scattered accounts. Entire-account compromise would be useful to target particular accounts, or to obtain a relatively complete picture of compromised accounts (e.g., for identity theft). In contrast, MDA-based attacks provide a direct way to compromise entire accounts.

### 3.1 Defining Non-Attributability

We define email non-attributability as a game involving an email protocol $\mathsf{E} = (\mathsf{Email}, \mathsf{VEmail})$, adversary $\mathscr{A}$, simulator $\mathscr{S}$, and distinguisher $\mathscr{D}$. An email protocol $\mathsf{E}$ is a pair of algorithms, run by the email sender $S$ and recipient(s) $R$ respectively. For an email server $S$ with internal state $s$,[6] $e \leftarrow \mathsf{Email}_s(S,R,m,\mu,t)$ denotes the information (bitstring) transmitted when $S$ sends $R$ an email with message $m$ and metadata $\mu$ at time $t$.[7] The recipient server $R$, upon receiving $e$, runs $\mathsf{VEmail}(e)$, which outputs a single bit indicating whether to accept the email as legitimate or reject it as spoofed.

Intuitively, we require indistinguishability between a legitimate email $e \leftarrow \mathsf{Email}_s(S,R,m,\mu,t)$ and a "fake" email that was created without access to the sending server at all. To model this, we consider a simulator $\mathscr{S}$ that "aims" to create such an email without $s$, and our security definition requires $e$ to be distributed indistinguishably from $\mathscr{S}$'s output.

This paper considers two definitions of non-attributability. *Recipient non-attributability* (Definition 5) considers a simulator that has access to a particular recipient's email server, and is required to output email from any sender to that recipient. $\Delta$-*universal non-attributability* (Definition 6) is an incomparable definition whose simulator is required to output email from any sender to any recipient while having access to neither email server. Formal definitions are in Appendix A.

**Relation to the threat models.** $\Delta$-universal non-attributability achieves non-attributability against after-the-fact attacks (Threat Model 1) for all emails sent and received at least $\Delta$ before the server is compromised.

Combining recipient non-attributability and $\hat{\Delta}$-universal non-attributability yields non-attributability against real-time attacks (Threat Model 2). A real-time attacker with ongoing access to an email server can easily make the fact of his access evident by immediately publishing all emails he sees (within time $\hat{\Delta}$ of receipt), but will be unable to convince third parties of any given email's authenticity since the fact of his access to the server allows him to forge emails in real time, under Definition 5. For allegedly compromised emails from more than $\hat{\Delta}$ ago, an attacker's credibility is even lower, since for such past timestamps *anyone with internet access* can generate seemingly validly signed emails, even without breaking into any email server at all, under Definition 6. The two definitions are complementary and incomparable.

**Necessity of recipient forgeries.** It may seem a counterintuitive or risky design choice to enable real-time email forgery in any part of the system. If forgery is restricted only to recipients forging emails to themselves, as in our definition, there is no spam/spoofing vulnerability — but given the choice, one might avoid introducing any forging capability at all, in the interest of a simpler and easier-to-analyze system. However, some sort of real-time forging capability by recipients is definitionally necessary to achieve non-attributability against real-time attacks: if the recipient cannot forge in real time, then any third party to whom a recipient server passes emails in real time must be convinced of the emails' authenticity.

**Other inherent model constraints.** A practical consequence of recipient non-attributability is that a recipient $R$'s email server can, *unknown to* $R$, create fraudulent messages that appear to be legitimate emails from any sender to $R$, and deliver them to $R$. As discussed §3, the current email system necessitates heavy client-server trust. In this context, recipient non-attributability does not meaningfully increase the trust a client places in her email server. For example, email servers in the current system could (and often do) omit DKIM headers when delivering emails to clients: this effectively implies the ability to deliver fake messages.

Also, we note that both definitions allow for strong, persistent attackers to convince others of the very fact that they have ongoing access to a particular email account. The definitions guarantee that even so, such attackers cannot make credible claims about email contents, since they gain the ability to falsify emails by the very fact of their access. That attackers with ongoing access can prove their access is unavoidable since universal forgeability is incompatible with spam resistance for too small $\Delta$, as discussed above.

**Adversarial secure hardware at recipient.** The requirement of spam- and spoofing-resistance means that any simulator $\mathscr{S}$ satisfying Definition 5 must use the recipient $R$'s secret state $r$: in order to prevent spam, real-time forgery must be limited to messages whose recipient is the forger herself. This suggests that recipient non-attributability would lose meaning in an extreme situation where every use of $r$ can be monitored and attested to, since then an attacker could prove that $\mathscr{S}$ was never invoked on $r$. This might be plausible assuming secure hardware, e.g., by generating and monitoring all uses of $r$ within a secure enclave (as suggested in [32]) — but even then, such an attack would likely only be feasible by the unlikely attacker who has designed her recipient email server with this unlikely configuration from its very setup. We note this possibility for completeness, but *such attacks are outside our threat models*, as mentioned earlier in §3.

**Malicious intermediaries and traffic logging.** Although our threat models focus on malicious recipient servers (as discussed earlier in §3), Definition 5 actually provides a meaningful, though limited, guarantee against malicious intermediaries (MTAs) as well. If a malicious MTA were to log all

[6] While this definition refers to "internal state $s$" for generality, the state $s$ essentially represents secret key material.

[7] Technically, $e$ may not be the string that $R$ eventually receives, as parties other than the sender (e.g., MTAs) routinely participate in email transmission and may influence the information en route. For simplicity, our notation glosses over this detail and uses $\mathsf{Email}_S(\cdots)$ to refer both to the string $S$ sends and the string $R$ receives. Also, this notation assumes that if an email has multiple recipients, each recipient receives the same information; this is true in the current email system but only some of our protocols. The possibility of different recipients receiving different information is elaborated in §5.3, and the notation can easily be tweaked to accommodate this, by treating $R$ as a tuple and having $\mathsf{Email}$ output a tuple of strings. For simplicity, however, we use the single-recipient notation for most of the exposition.

traffic and publish it in real time (perhaps even timestamped in a trustworthy way for future reference), in a system with immediate recipient forgeability, observers of the publications would still be unconvinced of: (1) whether any email the MTA claims is genuine (unforged) is really genuine, since the MTA could have *omitted* evidence of forgery, and (2) whether the MTA omitted any genuine emails from its publications.

**Why (sometimes) settle for weaker non-attributability?** KeyForge and TimeForge achieve only non-attributability against after-the-fact attacks, and their enhanced versions KeyForge$^+$ and TimeForge$^+$ are non-attributable against both after-the-fact and real-time attacks. Yet we consider KeyForge to be our main protocol and the most realistic proposal for deployment. In practice, the enhanced protocols' (unavoidable) interactivity and other overhead would often be compelling reasons to prefer the simpler base protocols except in contexts where addressing real-time attacks (or malicious intermediaries) is of heightened concern.

**Relation to deniability definitions in other contexts.** The cryptographic literature features many works on deniability of signatures and authentication, including (but not at all limited to) [25, 26, 36, 41, 45]. Our constructions could be seen as a practical instantiation of a deniable signature scheme subject to tight systems-based requirements.

## 4 Forward-Forgeable Signatures

Definition 1 formalizes *forward-forgeable signatures (FFS)*. They are a new primitive that this paper introduces, and are an essential building block for our proposed protocols. Informally, FFS are signature schemes equipped with a method to selectively "expire" past signatures by releasing *expiry information* that makes them forgeable. In an FFS, each signature is made with respect to a *tag* $\tau$, which is an arbitrary string (in our setting, a timestamp). Expiry information can be released with respect to any tag or set of tags. FFS have *correctness* and *unforgeability* requirements similar to standard signatures, as well as a new requirement, *forgeability on expiry*, that has no analogue in standard signatures.

**Definition 1** (FFS)**.** *A* forward-forgeable signature scheme (FFS) $\Sigma$ *is implicitly parametrized by message space* $\mathcal{M}$ *and tag space* $\mathcal{T}$*, and consists of five algorithms* $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Expire}, \mathsf{Forge})$.

SYNTAX:

- $\mathsf{KeyGen}(1^\kappa)$ *takes as input a security parameter*[8] $1^\kappa$ *and outputs a key pair* $(vk, sk)$.
- $\mathsf{Sign}(sk, \tau, m)$ *takes as input a signing key sk, a tag* $\tau \in \mathcal{T}$*, and a message* $m \in \mathcal{M}$*, and outputs a signature* $\sigma$.
- $\mathsf{Verify}(vk, \tau, m, \sigma)$ *takes as input a verification key vk, a tag* $\tau \in \mathcal{T}$*, a message* $m \in \mathcal{M}$*, and a signature* $\sigma$*, and outputs a single bit indicating whether or not* $\sigma$ *is a valid signature with respect to vk, m, and* $\tau$.
- $\mathsf{Expire}(sk, T)$ *takes as input a signing key sk and a tag set* $T \subseteq \mathcal{T}$*, and outputs expiry info* $\eta$.
- $\mathsf{Forge}(\eta, \tau, m)$ *takes as input expiry info* $\eta$*, a tag* $\tau \in \mathcal{T}$*, and a message* $m \in \mathcal{M}$*, and outputs signature* $\sigma$.

REQUIRED PROPERTIES:

1. Correctness *and* unforgeability *are straightforward adaptations of standard definitions. See Appendix B.*
2. Forgeability on expiry: *For all* $m \in \mathcal{M}, T \subseteq \mathcal{T}$*, for any* $\tau \in T$*, for any "distinguisher" algorithm* $\mathcal{D}$*, there is a negligible function* $\varepsilon$ *such that for all* $\kappa$,

$$\Pr\left[\begin{array}{ll} (vk,sk) \leftarrow \mathsf{KeyGen}(1^\kappa) & \\ \sigma_0 \leftarrow \mathsf{Sign}(sk,\tau,m) & \\ \eta \leftarrow \mathsf{Expire}(sk,T) & \\ \sigma_1 \leftarrow \mathsf{Forge}(\eta,\tau,m) & : \quad b = b' \\ b \leftarrow \{0,1\} & \\ b' \leftarrow \mathcal{D}(\sigma_b,\eta) & \end{array}\right] \leq 1/2 + \varepsilon(\kappa)\ .$$

*That is,* $\mathcal{D}$ *must not be able to distinguish whether a signature was produced using* $\mathsf{Sign}$ *or* $\mathsf{Forge}$*, even in the presence of the expiry information* $\eta$.

**Succinctness** The *succinctness* of an FFS is a measure of the efficiency of disclosure in terms of the size of expiry info per tag expired. Concretely, in our application, succinctness measures how expiry info scales as more non-attributable emails are exchanged over time. KeyForge uses a construction of FFS based on hierarchical identity-based signatures (§4.1), which achieves logarithmic succinctness.

**Definition 2.** *Let* $z : \mathbb{N} \to \mathbb{N}$*. Let* $S \subset \mathbb{P}(\mathcal{T})$ *be a set of sets of tags. A forward-forgeable signature scheme* $\Sigma$ *is* $(S, z)$-succinct *if for any* $T \in S$*, there is a negligible function* $\varepsilon$ *such that for all* $\kappa$,

$$\Pr_{(vk,sk) \leftarrow \mathsf{KeyGen}(1^\kappa)}\Big[\big|\mathsf{Expire}(sk,T)\big| \leq z(|T|)\Big] \geq 1 - \varepsilon(\kappa)\ .$$

### 4.1 FFS Construction from (Hierachical) IBS

We first outline a simple FFS construction BasicFFS based on identity-based signatures (IBS) [47], as a stepping stone to our main construction from hierarchical IBS (HIBS). The next paragraph assumes familiarity with standard IBS terminology; readers unfamiliar with IBS may skip ahead.

Let tags in the FFS correspond to identities in the IBS. BasicFFS.KeyGen outputs IBS master keys. The BasicFFS signing and verification algorithms for tag $\tau$ respectively invoke the IBS signing and verification algorithms for identity $\tau$. BasicFFS.Expire outputs the secret key for each input tag $\tau \in T$, and BasicFFS.Forge uses the appropriate secret key from the expiry information to invoke the IBS signing algorithm. This simple solution has linear succinctness. By leveraging hierarchical IBS (HIBS), our main construction achieves logarithmic succinctness, as described next.

[8] Technically, all five algorithms take $1^\kappa$ as an input, and $\mathcal{M}$ and $\mathcal{T}$ may be parametrized by $\kappa$. For brevity, we leave this implicit except in KeyGen.

**Definition 3.** *A* hierarchical identity-based signature scheme $\mathsf{HIBS}$ *is parametrized by message space $\mathcal{M}$ and identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in\mathbb{N}}$, and consists of four algorithms* $\mathsf{HIBS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\kappa)$ *takes as input a security parameter*[9] *and outputs a master key pair $(mvk, msk)$.*
- $\mathsf{KeyGen}(sk_{\vec{id}}, id)$ *takes as input a secret key $sk_{\vec{id}}$ for a tuple of identities $\vec{id} = (id_1,\dots,id_\ell) \in \mathcal{I}_1 \times\dots\times \mathcal{I}_\ell$ and an additional identity $id \in \mathcal{I}_{\ell+1}$ and outputs a signing key $sk_{\vec{id}'}$ where $\vec{id}' = (id_1,\dots,id_\ell,id)$. The tuple may be empty (i.e., $\ell = 0$): in this case, $sk_{()} = msk$.*
- $\mathsf{Sign}(sk_{\vec{id}}, m)$ *takes as input a signing key $sk_{\vec{id}}$ and a message $m \in \mathcal{M}$, and outputs a signature $\sigma$.*
- $\mathsf{Verify}(mvk, \vec{id}, m, \sigma)$ *takes as input master verification key mvk, tuple of identities $\vec{id}$, message $m \in \mathcal{M}$, and signature $\sigma$, and outputs a single bit indicating whether or not $\sigma$ is a valid signature with respect to mvk, $\vec{id}$, and m.*

*A* depth-$L$ $\mathsf{HIBS}$ *is a HIBS where the maximum length of identity tuples is L, i.e., the identity space is $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in[L]}$.*

Definition 3 establishes only syntax; for (standard) formal correctness and security definitions, see, e.g., [29].

**Definition 4.** *For an identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in\mathbb{N}}$, we say $\vec{id}$ is a* level-$\ell$ identity *if $\vec{id} \in \mathcal{I}_1 \times\dots\times \mathcal{I}_\ell$. For any $\ell' > \ell$, let $\vec{id}$ be a level-$\ell$ identity and $\vec{id}'$ be a level-$\ell'$ identity. We say that $\vec{id}'$ is a* sub-identity *of $\vec{id}$ if $\vec{id}$ is a prefix of $\vec{id}'$. If moreover $\ell' = \ell + 1$, we say $\vec{id}'$ is a* immediate sub-identity *of $\vec{id}$.*

**Deriving subkeys** Given a master secret key of a HIBS, it is possible to derive secret keys corresponding to level-$\ell$ identities for any $\ell$, by running $\mathsf{KeyGen}$ $\ell$ times. By a similar procedure, given any secret key corresponding to a level-$\ell$ identity $\vec{id}$, it is possible to derive any "subkeys" thereof, i.e., secret keys for sub-identities of $\vec{id}$. For our construction, it is useful to name this (simple) procedure: we define $\mathsf{HIBS.KeyGen}^\star$ in Algorithm 1. We write the randomness $\rho_1,\dots,\rho_\ell$ of $\mathsf{HIBS.KeyGen}^\star$ explicitly.

**Algorithm 1** $\mathsf{HIBS.KeyGen}^\star$

```
Input: $sk, \ell, \vec{id} = (id_1,\dots,id_{\ell'})$        ▷ Require: $\ell \le \ell'$
Randomness: $\rho_1,\dots,\rho_{\ell'}$
  for $j = \ell+1,\dots,\ell'$ do
      $sk \leftarrow \mathsf{HIBS.KeyGen}(sk, id_j; \rho_j)$
  return $sk$
```

**Succinctly representing expiry information** Given any set $T$ of tuples of identities, the simplest way to make signatures with respect to $T$ forgeable would be to release the secret key corresponding to each $\vec{id} \in T$, much as in $\mathsf{BasicFFS}$:

$$\eta = \left\{ sk_{\vec{id}} = \mathsf{HIBS.KeyGen}^\star(msk, 0, \vec{id}) \right\}_{\vec{id}\in T} . \qquad (1)$$

However, leveraging the hierarchical nature of HIBS, $\eta$ can often be represented more succinctly than (1). Based on the fact that Algorithm 1 allows the derivation of any subkey, we make two optimizations. First, before computing (1), we delete from $T$ any $\vec{id} \in T$ that is a sub-identity of some $\vec{id}' \in T$. Secondly, if there is any $\vec{id}' = (id_1,\dots,id_\ell) \in \mathcal{I}_1 \times\dots\times \mathcal{I}_\ell$ such that every immediate subkey of $\vec{id}'$ is in $T$ (i.e., $\forall id_{\ell+1} \in \mathcal{I}_{\ell+1}$, $(id_1,\dots,id_\ell,id_{\ell+1}) \in T$), then all sub-identities of $id'$ can be removed from $T$ and replaced by $id'$ before computing (1). Such replacement is permissible only when *every* possible subkey of $id'$ is derivable from $T$: otherwise, adding $id'$ to $T$ would implicate additional subkeys outside $T$.

These two optimizations yield an algorithm $\mathsf{Compress}$, which takes as input a set of identity tuples $T$, and outputs a (weakly) smaller set of identity tuples $T'$ such that knowledge of the secret keys corresponding to $T'$ enables computing valid signatures with respect to exactly the identity tuples in $T$. HIBS security guarantees that even given $T'$, signatures for identity tuples not in $T$ remain unforgeable. Next, we describe how $\mathsf{Compress}$ works using a tree-based representation of identity tuples (a formal specification is given in the full version [48]).

**Tree representation** It is convenient to think of identity tuples represented graphically in a tree. A node at depth $\ell$ represents a tuple of $\ell$ identities (the root node is depth 0). The set of all depth-$\ell$ nodes corresponds to the set of all $\ell$-tuples of identities. The branching factor at level $\ell$ is $|\mathcal{I}_{\ell+1}|$. Given a secret key for a particular node (i.e., identity tuple), the secret keys of all its descendant nodes are easily computable using $\mathsf{HIBS.KeyGen}^\star$. (The secret key for the root node is the master secret key.) In this language, $\mathsf{Compress}$ simply takes a set $T$ of nodes and returns the smallest set $T'$ of nodes such that (1) all nodes in $T$ are descendants of some node in $T'$ and (2) no node not in $T$ is a descendant of any node in $T'$.

Our construction of FFS based on HIBS follows.

**Construction 1.** *Let* $\mathsf{HIBS}$ *be a depth-L HIBS*[10] *with message space $\mathcal{M}$ and identity space $\mathcal{I} = \{\mathcal{I}_\ell\}_{\ell\in[L]}$. Let $\mathcal{O}$ be a random oracle,*[11] *and for any tuple $\vec{\tau} = (\tau_1,\dots,\tau_\ell)$, let $\vec{\mathcal{O}}(\vec{\tau}) = (\mathcal{O}(\tau_1),\dots,\mathcal{O}(\tau_\ell))$. For $\ell \in [L]$, define $\mathcal{T}_\ell = \mathcal{I}_1 \times\dots\times \mathcal{I}_\ell$. We construct a FFS $\Sigma$ with message space $\mathcal{M}$ and tag space $\mathcal{T} = \bigcup_{\ell\in[L]} \mathcal{T}_\ell$, as follows.*

- $\Sigma.\mathsf{KeyGen}(1^\kappa)$*: output* $(vk, sk) \leftarrow \mathsf{HIBS.Setup}(1^\kappa)$.
- $\Sigma.\mathsf{Sign}(sk, \vec{\tau} = (\tau_1,\dots,\tau_\ell), m)$*:*
  *let* $sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^\star(sk, 0, \vec{\tau}; \vec{\mathcal{O}}(\vec{\tau}))$
  *and output* $\sigma \leftarrow \mathsf{HIBS.Sign}(sk_{\vec{\tau}}, m)$.
- $\Sigma.\mathsf{Verify}(vk, \vec{\tau}, m, \sigma)$*: output* $b \leftarrow \mathsf{HIBS.Verify}(vk, \vec{\tau}, m, \sigma)$.
- $\Sigma.\mathsf{Expire}(sk, T)$*: let* $T' = \mathsf{Compress}(\mathcal{I}, T)$*; output*

$$\eta = \left\{ (\vec{\tau}, sk_{\vec{\tau}}) : sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^\star(sk, 0, \vec{\tau}; \vec{\mathcal{O}}(\vec{\tau})) \right\}_{\tau\in T'} .$$

[9] Technically, all four algorithms take $1^\kappa$ as an input, and $\mathcal{M}$ and $\mathcal{I}$ may be parametrized by $\kappa$. For brevity, we leave this implicit except in $\mathsf{Setup}$.

[10] The depth need not be finite, but we consider finite $L$ for simplicity.

[11] The construction is presented in the random oracle model for simplicity, but does not *require* a random oracle: the random oracle can be replaced straightforwardly by a pseudorandom function (PRF) where the PRF key is made part of the HIBS secret key.

- $\Sigma.\mathsf{Forge}(\eta, \tau, m)$: *if there exists* $sk_{\tau'}$ *such that* $(\tau', sk_{\tau'}) \in \eta$ *and* $\tau'$ *is a prefix of* $\tau$, *let* $\ell$ *be the length of* $\tau'$, *let* $sk_{\vec{\tau}} = \mathsf{HIBS.KeyGen}^{\star}(sk_{\tau'}, \ell, \vec{\tau}; \vec{\mathscr{O}}(\vec{\tau}))$ *and output* $\sigma \leftarrow \mathsf{HIBS.Sign}(sk_{\vec{\tau}}, m)$; *otherwise, output* $\perp$.

**Theorem 1.** *If* HIBS *is a secure HIBS, Construction 1 instantiated with* HIBS *is a FFS with logarithmic succinctness for sequentially ordered tag expiry. (Formal statement and proof is in the full version [48].)*

**Discussion of alternative approaches** Forward-*secure* signatures (FSS) bear some resemblance to FFS, but have a different goal: namely, enabling efficient key updating while preventing derivation of *past* keys from present and future keys. In contrast, our setting requires that *present and future* keys cannot be derived from past keys. (See Appendix C for more detailed comparison.) One could build a FFS from a FSS by computing a long list of secret keys and then using them *in backwards order*. Using techniques of [23, 35], a sequence of keys could moreover be stored with logarithmic storage and computation to access a key. However, this optimization is only designed for contiguous sequences of keys; HIBS-based schemes allow for some succinct non-sequential key release and thus support more nuanced tag structures. Still, for certain applications, e.g., postquantum sequential key release, an FFS based on a FSS such as XMSS [13] could be useful.

The requirements of FFS also have some similarity to *timed authentication*. The TESLA timed authentication protocol [43, 44] considers releasing authentication (MAC) keys following a delay after sending the payload, in the broadcast authentication context. Such delayed verification is untenable for email for several reasons, even beyond the inconvenience of waiting 15 minutes for email delivery. Email's store-and-forward nature (see §2.2) means multiple MTAs may need to verify emails before forwarding (e.g., for spam filtering): if the first MTA waits to verify before forwarding, the next MTA will be unable to verify because the delay has rendered the authentication forgeable. Also, the inability to discard incoming spam before a time delay may increase denial-of-service vulnerability, especially for smaller email providers.

## 5 Our Protocol Proposals

### 5.1 KeyForge

KeyForge consists of two components: (1) replace the digital signature scheme used in DKIM with a succinct FFS; and (2) email servers periodically publish expiry information. In this section, we assume all algorithms have access to a global publication mechanism or bulletin board (as noted in §3).

**FFS configuration for KeyForge.** Figure 2 illustrates KeyForge's key hierarchy. KeyForge is based on an $L$-level tag structure, corresponding to identity space $\mathscr{I} = \{\mathscr{I}_\ell\}_{\ell \in [L]}$ where the level-$L$ identities represent 15-minute time chunks spanning a 2-year period. We use the following intuitive 4-level configuration for ease of exposition, but as discussed in §6, it is preferable for efficiency to keep $|\mathscr{I}_\ell|$ equal for all $\ell \in [L]$.

| | |
|---|---|
| $\mathscr{I}_1 = \{1, 2\}$ | representing a 2-year time span |
| $\mathscr{I}_2 = \{1, \ldots, 12\}$ | representing months in a year |
| $\mathscr{I}_3 = \{1, \ldots, 31\}$ | representing days in a month |
| $\mathscr{I}_4 = \{1, \ldots, 96\}$ | representing 15-minute chunks of a day |

A tag $\tau = (\mathsf{y}, \mathsf{m}, \mathsf{d}, \mathsf{c}) \in \mathscr{I}_1 \times \mathscr{I}_2 \times \mathscr{I}_3 \times \mathscr{I}_4$ corresponds to a 15-minute chunk of time. The 15-minute chunks are contiguous, consecutive, and disjoint, so that any given timestamp is contained in exactly one chunk. $\tau(t)$ denotes the unique 4-tuple tag $(\mathsf{y}, \mathsf{m}, \mathsf{d}, \mathsf{c})$ that represents the chunk of time containing a timestamp $t$, and $t \sqsubset \tau$ denotes that $\tau$ represents a chunk of time containing timestamp $t$.

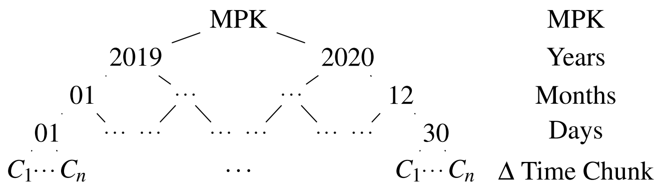

Figure 2: KeyForge Hierarchy Layout

KeyForge requires each signature at time $t$ to be with respect to a tag (timestamp) $t + \hat{\Delta}$. The tag is sent in the email's header, and used for verification at the receiving server. Algorithm 2 specifies KeyForge's signing and verification (key generation is identical to that of the underlying FFS).

**Algorithm 2** KeyForge.Sign and KeyForge.Verify

```
$t = \mathsf{CurrentTime}()$
function KeyForge.Sign($sk, m, \Delta$)
    return $(\tau(t+\Delta), \sigma \leftarrow \mathsf{FFS.Sign}(sk, \tau(t+\Delta), m))$
function KeyForge.Verify($vk, m, \tau, \sigma$)
    return $t \sqsubset \tau$ AND $\mathsf{FFS.Verify}(vk, \tau, m, \sigma)$
```

Then we build an email protocol $\mathsf{E}_{\mathsf{KF}} = (\mathsf{Email}, \mathsf{VEmail})$ as follows. Let $(vk, sk)$ be the key pair of a sending server. $\mathsf{Email}_{sk}(S, R, m, \mu, t)$ outputs $(\zeta, \tau_\zeta, \sigma_\zeta)$ where $\zeta = (S, R, m, \mu)$, $(\tau_\zeta, \sigma_\zeta) \leftarrow \mathsf{KeyForge.Sign}_t(sk, \zeta, \hat{\Delta})$, and the subscript $t$ denotes an execution of KeyForge.Sign at time $t$. $\mathsf{VEmail}(\zeta, \tau_\zeta, \sigma_\zeta)$ runs $\mathsf{KeyForge.Verify}(vk, \zeta, \tau_\zeta, \sigma_\zeta)$ and outputs the result (where the recipient obtains $vk$ by looking up $S$'s key in DNS).

**Efficient tree regeneration from private keys.** A key feature of our FFS construction is that the private keys from children (e.g., day-keys) are easy to generate from parent keys (e.g., the MSK). This is not implied by the definition of

HIBS,[12] and is essential for succinct expiry of entire portions of the tree (e.g., a year) by disseminating a single key. Further, regeneration can enhance security and availability: to limit key exposure, organizations could store the MSK in an HSM disconnected from the Internet, and keep only a child key pair in the MSA, thereby mitigating damage in case of compromise and allowing recovery from failure.

**Where to publish expiry information?** Regeneration allows KeyForge to have succinct expiry information; the number of private keys necessary to represent all expired chunks depends on the tree's structure (see §6), but amounts to less than 4 KB for reasonable configurations. In contrast, the analogous construction based on non-hierarchical IBS would have expiry information growing linearly throughout a (two-year) master-key lifespan, resulting in megabytes of expiry information.

Small expiry information means ease of distribution. While our implementation uses a simple public-facing webserver, one could imagine posting via DNS TXT records, public blockchains, or in outgoing email headers. Slow but permanent techniques (e.g., a blockchain) for keys higher up in the hierarchy (e.g., a year) could ensure that such keys are permanently available.

**When to publish expiry information?** KeyForge requires email servers to publish expiry information at regular intervals. A natural option is to publish expiry information every 15 minutes; to publish the expiry information corresponding to each chunk $\mathsf{c}$ at the end of the time period that $\mathsf{c}$ represents.

Publishing every 15 minutes yields the finest granularity of expiry possible under the basic four-level tag structure. Based on a server's preference, it could release information at longer intervals (e.g., days) or shorter ones. In case of an attack, an adversary would be able to convince third parties of the authenticity of all emails in the current interval (e.g., the current day), so risk aversion prefers shorter intervals.

Server misconfiguration and clock skew may cause minor clock discrepancies between the MSA and MDA. To account for this, we delay publishing "expired" keys by 5 minutes. Although in practice most emails are received very quickly, the SMTP RFC [37] has a very lax give-up time of 4 days. To get a rough idea of how quickly emails tend to be delivered, we computed the time differences from the first Received header to the last in the Podesta email corpus [56],[13] and found that, of the 48,246 messages with parseable Received timestamps, over 99% (47,349) took less than 12 minutes.

While expiry time is a configurable parameter of KeyForge (e.g., by administrators), keeping it short is advisable to minimize time until universal forgeability. We leave a detailed study of email delivery times in practice to future work, while noting that such a study might support considerably reducing our conservative 15 minutes, and/or tailoring our approach to specific delay-prone situations. For example, delays are often caused by expected receiving-server outages (e.g., for server updates), which might be resolved by using a DMARC-like DNS record to signal to the sender to hold messages until later. Anti-spam techniques such as greylisting can delay email by 15 minutes more than usual; to address this, we can add 17 minutes' leeway when first sending to a new domain.

We do not fully detail remediation procedures for timeouts, but note that similar authentication failures happen under DKIM and are commonly resolved via feedback loops such as Authentication Failure Reporting [27]. Shortening our expiry time is tricky given potentially adversarial routing delays: providing TCP-like flow control would be systematically possible, but we should also account for malicious MTAs trying to prolong messages' unforgeability. A hard-cutoff maximum would likely be advisable.

**Why 15-minute chunks?** The time period associated with each leaf node is the maximum granularity of expiry information release. $\hat{\Delta}$ is a lower bound on chunk size: since $\hat{\Delta}$ represents email delivery time, publishing expiry information more often does not make sense.

**Why a 2-year public key lifetime?** Rotating keys is good practice; for operational reasons, the M3AAWG recommends DKIM key rotation every 6 months [39]. However, recognizing that, realistically, DKIM keys often last more than 6 months, our evaluation assumes a 2-year period.

**How many levels?** The optimal $L$ depends on a trade-off between computation time and expiry succinctness; see §6.

**Flexible expiry policies** The basic tag structure described above is customizable: e.g., an extra level $\mathscr{I}_\ell$ might represent an email's "sensitivity," allowing sensitive emails to expire faster. Alternatively, one might want certain emails to expire more slowly or never (e.g., bank/employer emails or contracts). KeyForge is highly configurable: after the first four levels, different email servers' policies need not be consistent.

## 5.2 TimeForge

KeyForge's main limitation is that it requires signers to continuously release key material. Wide distribution can pose a practical challenge; users must depend on their provider to perform this task reliably. Unreliable distribution would limit a system's realistic deniability.

TimeForge takes a different approach that eliminates reliance on follow-up action by signers. TimeForge leverages a *publicly verifiable timekeeping service* (PVTK), as defined in §3. In this section, all algorithms are assumed to have access to a common PVTK. A PVTK could be realized using various extant Internet systems, as discussed in more detail in the full version of this work [48].

The intuition behind TimeForge is straightforward. Let $M$ be an email message sent at time period $t$. The sender first

[12] The definition of HIBS is compatible with this property, but does not require it. Constructions typically have randomized subkey generation processes so do not have reproducible child keys.

[13] Beyond the irony, we chose the Podesta email corpus as it was distributed intact with attachments, and thus arguably more representative of a realistic user's email distribution than other public datasets.

signs each message using a standard SUF-CMA signature scheme to produce a signature $\sigma$. She then authenticates the message, not directly using $\sigma$, but rather using a witness indistinguishable and non-interactive proof-of-knowledge (PoK) of the (informal) statement: I know a valid sender signature $\sigma$ on $M$ OR I know a valid PVTK proof $\pi_{t+d}$, for some $d \geq \Delta$.

Assuming a trustworthy PVTK service, this proof authenticates the message during any time period prior to $t+\Delta$. Once a PVTK proof $\pi_{t+\Delta}$ becomes public, the PoK becomes trivial for any party to generate. Witness indistinguishability ensures that a signer's valid proof is indistinguishable from a "forgery" later computed using a revealed PVTK proof.

**PVTK.** A PVTK scheme comprises three algorithms.

- $\mathsf{TK.Setup}(1^\lambda)$ takes a security parameter $\lambda$ and outputs a set of public parameters *params* and a trapdoor *sk*.
- $\mathsf{TK.Prove}(sk,t)$ takes as input *sk* and the current time epoch $t$, and outputs a proof $\pi_t$.
- $\mathsf{TK.Verify}(params,t,\pi_t)$ on input *params*, a time period $t$, and the proof $\pi_t$, outputs whether $\pi_t$ is valid.

*Correctness and Security.* Correctness is straightforward. $\Delta$-*PVTK security* requires that an adversary with a PVTK oracle must not be able to produce a valid proof for some time period $t_{\mathsf{max}}+\Delta$ (except with negligible probability) where $t_{\mathsf{max}}$ is the largest oracle query, and $\Delta > 0$ is a constant parameter.

**Realizing a PVTK service.** A simple PVTK system can be constructed using a single server that maintains a clock, and periodically signs the current time using an SUF-CMA signature (our implementation does this). While conceptually simple, deploying this solution at scale is likely to be costly, and may suffer denial-of-service and network-based attacks. A better approach might construct a PVTK from *existing* Internet services: in the full version of this work [48] we outline proposals based on OCSP servers, certificate transparency, randomness beacons, proof-of-work-based blockchains, and verifiable delay functions.

**A basic TimeForge signature scheme.** The TimeForge scheme consists of four algorithms: $\mathsf{TF.Keygen}$, $\mathsf{TF.Sign}$ and $\mathsf{TF.Verify}$, and $\mathsf{TF.Forge}$. We assume a PVTK scheme with parameters *params* and an SUF-CMA signing algorithm $\mathsf{Sig}$.

- $\mathsf{TF.Keygen}(1^\lambda, params)$. Run $\mathsf{Sig.Keygen}(1^\lambda)$ to generate $(pk,sk)$ and output $PK=(pk,params)$, and $SK=sk$.
- $\mathsf{TF.Sign}(PK,SK,M,t,\Delta)$. Parse $PK=(pk,params)$. On input a message $M$ and a time period $t$, compute $\sigma \leftarrow \mathsf{Sig.Sign}(SK,M\|t\|\Delta)$ and the following WIPoK:[14]

$$\Pi = \mathsf{NIPoK}\{(\sigma,s,\pi) : \mathsf{Sig.Verify}(pk,\sigma,M\|t\|\Delta)=1 \ \vee \ (\mathsf{TK.Verify}(params,\pi,s)=1 \ \wedge \ s \geq t+\Delta)\}$$

  Output $\sigma_{\mathsf{tf}} = (\Pi,t,\Delta)$.
- $\mathsf{TF.Verify}(PK,M,\sigma_{\mathsf{tf}})$. Parse $PK=(pk,params)$ and $\sigma_{\mathsf{tf}}=(\Pi,t,\Delta)$, verify the proof $\Pi$ w.r.t. public values $t,\Delta,pk,M$, and output the verification result.

$\mathsf{TF.Forge}$ takes as input a PVTK proof $\pi_s$ for $s \geq t+\Delta$.

- $\mathsf{TF.Forge}(PK,M,t,s,\Delta,\pi_s)$. parse $PK=(pk,params)$ and compute the NIPoK $\Pi$ described in the $\mathsf{TF.Sign}$ algorithm, using the witness $(\perp,s,\pi_s)$. Output $\sigma_{\mathsf{tf}}=(\Pi,t,\Delta)$.

Now we can define $\mathsf{E_{TF}}$ analogously to $\mathsf{E_{KF}}$ from §5.1. Let us define the TimeForge email protocol $\mathsf{E_{TF}} = (\mathsf{Email},\mathsf{VEmail})$ as follows. Let $(PK,SK)$ be the key pair of a sending server. $\mathsf{Email}_{SK}(S,R,m,\mu,t)$ outputs $(\zeta,\sigma_\zeta)$ where $\zeta=(S,R,m,\mu)$ and $\sigma_\zeta \leftarrow \mathsf{TF.Sign}(PK,SK,\zeta,t,\hat{\Delta})$. $\mathsf{VEmail}(\zeta,\sigma_\zeta)$ runs $\mathsf{TF.Verify}(PK,\zeta,\sigma_\zeta)$ and outputs the result (where the recipient obtains $PK$ by looking up $S$'s key in DNS.) Appendix D discusses possible concrete constructions of TimeForge.

## 5.3 KeyForge$^+$ and TimeForge$^+$

KeyForge$^+$ (resp. TimeForge$^+$) consists of KeyForge (resp. TimeForge) with two modifications: a *forge-on-request protocol* and *per-recipient-domain signatures*, described next.

**1. Forge-on-request protocol.** We add a protocol $\mathsf{F}$ (detailed in Algorithm 3) by which an email server $S$ accepts real-time requests for specified email content to be sent to the requester (and nobody else). We write $A^{\mathsf{F}}$ to denote that an algorithm $A$ has access to email forgeries via $\mathsf{F}$. The forge-on-request protocol ensures that all users have the capability to forge emails to themselves in real time, directly achieving *immediate recipient forgeability*. The requirement that the recipient be the requester is crucial: each requester is enabled to forge emails *only to herself.*

The requester's email server attests to the requesting client's identity (similarly to DKIM). We note that a malicious server could unauthorizedly sign requests for any client account it controls. This is outside our threat model, and such behavior is equally possible under DKIM (see also "Client-server trust" under §3): that is, today's email ecosystem already relies on servers to attest honestly to their clients' identities, and allows servers to spam their own clients (a behavior that might not keep them many clients).

**2. Per-recipient-domain signatures.** In KeyForge$^+$ and TimeForge$^+$, the MSA signs each outgoing email *once per recipient domain*: producing a signature $\sigma_D \leftarrow \mathsf{Sign}(sk,(D,m))$ for each recipient domain $D$, where $sk$ is the signing key and $m$ is the email data that the sending server would have signed under DKIM (or KeyForge or TimeForge). Then, it sends each recipient domain $D$ the email and (only) $\sigma_D$. Per-recipient-domain signatures prevent attackers from using the forge-on-request protocol to send spam/spoofing emails to co-recipients on forged emails. Adida *et al.* [2] previously proposed per-recipient signatures in a very similar context.

We define the email protocols $\mathsf{E_{KF+}}$ and $\mathsf{E_{TF+}}$ accordingly. $\mathsf{E_{KF+}.Email}(S,(R_1,\ldots,R_N),m,\mu,t)$ outputs $(e_1,\ldots,e_N)$ where $e_i \leftarrow \mathsf{E_{KF}.Email}(S,R_i,m,\mu,t)$; and $\mathsf{E_{KF+}.VEmail}$ is just as in $\mathsf{E_{KF}}$. $\mathsf{E_{TF+}}$ is defined analogously, w.r.t. $\mathsf{E_{TF}}$.

[14] Here we use Camenisch-Stadler notation, where the witness values are in parentheses () and any remaining values are assumed to be public.

| Monthly KeyForge$_B$ | KeyForge$_B$ $\sigma$ | Monthly KeyForge$_C$ | KeyForge$_C$ $\sigma$ | DKIM RSA2048 $\sigma$ | TimeForge $\sigma$ |
|---|---|---|---|---|---|
| $30 \times 65 = 1950$ | 98 | $30 \times (64+32) = 2880$ | $64 \times 2 + 32 = 160$ | 256 | 841 |

Table 1: Bandwidth costs (in bytes) of KeyForge$_B$, KeyForge$_C$, and DKIM with RSA. $\sigma$ denotes a signature.

**Theorem 2.** *$\mathsf{E}_{\mathsf{KF}}$ and $\mathsf{E}_{\mathsf{KF+}}$ are $\hat{\Delta}$-universally non-attributable (Definition 6). Assuming email servers adopt the forge-on-request protocol $\mathsf{F}$, $\mathsf{E}_{\mathsf{KF+}}$ is further non-attributable for recipients (Definition 5). (Proof is in the full version [48] due to space constraints.)*

**Theorem 3.** *Assuming a PVTK, $\mathsf{E}_{\mathsf{TF}}$ and $\mathsf{E}_{\mathsf{TF+}}$ are $\hat{\Delta}$-universally non-attributable and $\mathsf{E}_{\mathsf{TF+}}$ is further non-attributable for recipients.*

**On the efficiency of KeyForge$^+$/TimeForge$^+$** Per-recipient-domain signatures add sender-side (but not receiver-side) overhead compared to schemes like DKIM, KeyForge, or TimeForge. While the overhead is unlikely to be prohibitive given the efficiency of signing, it must be taken into account when evaluating KeyForge$^+$ and TimeForge$^+$ (see Section 6). Implementing forge-on-request and per-recipient-domain signatures would entail more complexity and significant changes to the existing email infrastructure, than the base protocols. While immediate recipient forgeability is desirable for added protection against real-time attacks (see Threat Model 2), KeyForge is a more realistic candidate for near-term deployment as it is realizable with lighter-weight changes to the existing system: namely, replacing DKIM's signature scheme, and unilateral server publication of small amounts of data.

**Notation** $\mathsf{Email}_s(S,R,m,\mu,t)$ is as defined in §3.1, additionally taking into account that signatures in KeyForge$^+$ and TimeForge$^+$ are per recipient domain. $\mathsf{FReq}$ denotes a special message to betoken forge requests. For an email address $a$, let $a$.dom denote its domain.

## 6 Implementation and Evaluation

We implemented prototypes of KeyForge and TimeForge and integrated them into Postfix, a common MDA/MSA. Our code is open source [1]. We performed all benchmarks on a 2017 MacBook Pro, 15-inch, with an Intel 4-core 3.1GHz processor and 16GB of RAM. We use the RELIC toolkit's [4] implementation of a BN-254 curve. This configuration conservatively yields keys with a 110-bit security level [5], which is on par with the standard 2048-bit RSA. We chose RELIC due to its support for many pairing friendly curves and low overhead.

**Algorithm 3** Forge-on-request protocol $\mathsf{F}$

```
Requester (client)
To request an email with message $m$ and metadata $\mu$ from alice@foo.com:
  • Send $(\mathsf{FReq}, m, \mu, \texttt{alice@foo.com})$ to client's (i.e., its own) email server.
Email server (say, bar.com, with secret key $s$)
On receiving request $(\mathsf{FReq}, m, \mu, a)$ from own client bob:[15]
  • If $a$.dom = bar.com:[16] Let $t$ be the current time. Deliver $e$ to bob, where $e \leftarrow \mathsf{Email}_s(a, \texttt{bob@bar.com}, m, \mu, t)$.
  • Else: Let $\sigma \leftarrow \mathsf{Sign}(\mathsf{FReq}, m, \mu, a, \texttt{bob})$.[17] Send $(\mathsf{FReq}, m, \mu, a, \texttt{bob}, \sigma)$ to server $a$.dom.
On receiving request $(\mathsf{FReq}, m, \mu, a, b, \sigma)$ from server $b$.dom:
  • $v \leftarrow \mathsf{Verify}(vk, (\mathsf{FReq}, m, \mu, a, b), \sigma)$, where $pk$ is $b$.dom's public key in DNS.
  • If $v = 0$: Do not respond.
  • Else (i.e., $v = 1$): Let $t$ be the current time. Send $e, e'$ to $b$.dom, where $e \leftarrow \mathsf{Email}_s(a, b, m, \mu, t)$ and $e' \leftarrow \mathsf{Email}_s(a, b, m, \mu, t - \hat{\Delta})$.
```

We evaluate two versions of KeyForge instantiated with different HIBS schemes: (1) KeyForge$_B$, which uses Gentry-Silverberg's "BasicHIDE" bilinear map based scheme [29] using a BN254 curve and (2) KeyForge$_C$, which uses certificate chains on public keys using non-identity-based signatures, instantiated with Ed25519.[18] We also implemented a prototype of TimeForge (see Appendix D), which is less efficient; it is intended as a proof of concept whose practicality will improve with advances in the underlying proof primitives (an active area of research). The two KeyForge implementations share the following bandwidth optimization.

**KeyForge bandwidth optimization.** HIBS schemes tend to have relatively large signatures. In KeyForge$_B$, a signature must include public parameters for each node on the path to the current chunk. A public parameter in this configuration is 65B, yeilding a bandwidth of 260B for a four-level Y/M/D/Chunk tree, resulting in a total of 293B per signature. KeyForge$_C$ similarly requires an Ed25519 signature between each node in the hierarchy, and has total signature size of 448B (four 64B path signatures, four 32B public keys, and the message signature). We optimize bandwidth by precomputing all path parameters except for the last chunk and store them in the DNS, along with the MPK. When verifying from a new server, KeyForge performs a DNS lookup and caches the result at a cost of 2-3KB per month (see Table 1).

[15] We assume client-server communication is authenticated.

[16] I.e., if the request is for a forgery from another address in the requester's own domain.

[17] $\mathsf{Sign}$ denotes the signing algorithm of any secure signature scheme.

[18] The certificate-based approach has been attributed to folklore.

|  | Sign(ms) | Sign/s | Verify(ms) | Verify/s |
|---|---|---|---|---|
| TimeForge | 24.58 | 49.68 | 23.24 | 43 |
| KeyForge$_B$ | 0.34 | 2,932 | 3.36 | 298 |
| KeyForge$_C$ | 0.13 | 17,197 | 0.13 | 7,541 |
| RSA2048 | 0.93 | 1,075 | 0.05 | 19,966 |
| Ed25519 | 0.03 | 27,001 | 0.10 | 9,781 |

Table 2: Time required for a single operation in milliseconds, and the equivalent number of operations per second.

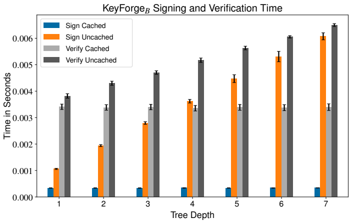

Figure 3: KeyForge$_B$ timings

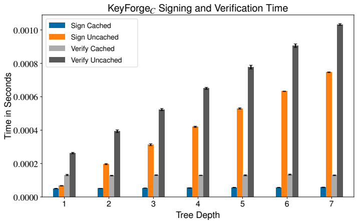

Figure 4: KeyForge$_C$ timings

Two components, the keyserver and mail filter, are shared between all implementations. They are described next.

**Mail Filter.** The filter ensures that sent emails are properly formatted, verifies incoming emails, and communicates the results to the MDA/MTA. The filter works by intercepting SMTP requests, adding necessary metadata to outgoing email headers, and requesting cryptographic operations from the keyserver. When sending a message, the filter attaches an expiry time (and other verification information) to the email's header, hashes the metadata and message content, forwards the hash to the keyserver to sign, and finally adds this signature to the header. On receipt, the filter confirms that the signature's hash matches the message and metadata, and forwards the signature, sending domain, and expiry timestamp to the keyserver for verification. If verification fails, the filter alerts PostFix and the message is dropped.

**Keyserver.** The keyserver performs signing and verification, communicates with the mail filter over RPC, and publishes expired keys (for KeyForge) via a simple webserver.

## 6.1 Evaluation

We evaluate messaging bandwidth, expiry data bandwidth, and speed. Our primary focus is on comparison with RSA-2048: it is the signature scheme commonly used in DKIM, and so a natural benchmark for practicality in the current email ecosystem. Although more bandwidth-efficient algorithms were approved for DKIM use some months ago, (e.g., Ed25519 with a 64 B signature [34]), these schemes appear to have had limited deployment to date.[19] Nonetheless, for completeness, this section also considers Ed25519 performance.

**Bandwidth.** Table 1 shows bandwidth costs for various configurations of KeyForge and TimeForge. Both KeyForge implementations have a bandwidth per email that is 42% *smaller* than a DKIM RSA-2048 signature.

**Speed.** To capture the range of KeyForge's possible performance, we considered two cases: (1) where the public key path is verified from scratch (e.g., in setting up a new server, or verifying messages from a new domain) and (2) where path parameters are pre-verified and cached. Figures 3 and 4 show the results. Signing is largely unaffected by tree depth when caching. Table 2 provides efficiency microbenchmarks for KeyForge, TimeForge, and Ed25519 and DKIM's RSA-2048 via the OpenSSL suite's benchmark. All KeyForge benchmarks are for a 4-level tree with caching. Note that experiments were run on a laptop with power lower than a common server, so our timings may be seen as upper bounds. Performance scales linearly with the number of cores; our measurements are for a single core.

**Optimizing for KeyForge expiration bandwidth.** While the Y/M/D/Chunk configuration is easy to intuit, an equal branching factor across tree levels yields a large gain in succinctness. For example, the average size of expiry info of trees with an equal branching factor for a 2-year period is 4.5MB, 4KB, or 1.8KB for depths 1, 4, and 7.

**Discussion and analysis.** We find that KeyForge, especially KeyForge$_C$, is likely practical when using DKIM's RSA-2048 as a benchmark. In both implementations, KeyForge's signing time is *better* than RSA: KeyForge$_B$ and KeyForge$_C$ sign 2.7 and 16 times faster than RSA, respectively. KeyForge further beats RSA on signature bandwidth per email, at just 63% or less of RSA signature size in the worst case. RSA outperforms KeyForge only on verification time: KeyForge$_C$ is still eminently practical, with verification a factor of two slower than RSA, whereas KeyForge$_B$ is an order of magnitude slower.

Verification time is unlikely to affect KeyForge's viability, as other factors such as hashing, I/O, and network latency are likely to dominate. Any hash-and-sign scheme must read the message into memory and perform a hash, so to provide a ballpark measurement of I/O and hashing, we timed OpenSSL's SHA256 on the Podesta corpus [56], stored on-disk. The average time required was 10.2ms (2.689ms std),[20] indicating that hashing and I/O is surprisingly impactful. Network latency is significant as well — SMTP requires that a sending MTA perform a *minimum* of four round trips per email.[21] A highly optimistic round-trip time of 5ms would yield of 20ms per email, not including time to send message content.

The choice between KeyForge$_B$ and KeyForge$_C$ is likely implementation dependent: while KeyForge$_B$ requires less

[19] E.g., as of October 2019, Gmail and Exchange use only RSA-2048.

[20] Email size is often pushed up by HTML formatting, embedded media, and attachments. Average email size in our corpus is 98 KB (691 KB std).

[21] SMTP messages require a round trip per command, and each email requires a MAIL, RCPT, and two DATA commands.

bandwidth, its drawbacks are speed and use of non-IETF-standardized curves (unlike KeyForge$_C$).

**A note on adoption.** With an ecosystem as unwieldy as email, a reasonable concern might be that any large-scale update would be difficult. That said, now is an opportune time to propose such changes: the IETF has recently approved a new standard that will encourage MTAs to begin updating their DKIM signing and verification algorithms [34]. Further, if the community were to endorse a new standard, one could imagine large email providers (e.g., Google) displaying favorable security indicators akin to to Gmail's TLS indicators [30]. Such tactics have been successful in the context of HTTPS.

We have consulted members of the IETF, W3C, and the Gmail Security team, and optimized and evaluated our prototypes with their performance priorities and concerns in mind.

## Acknowledgements

We are grateful to Jon Callas for helpful discussions about motivations for email non-attributability and our scheme's applicability to DKIM, and to Dan Boneh, Daniel J. Weitzner, John Hess, Bradley Sturt, Stuart Babcock, and Ran Canetti for their feedback on earlier versions of this work. This work was supported in part by the William and Flora Hewlett Foundation grant 2014-1601, and by the MIT Media Lab's Digital Currency Initiative and its funders. We would like to acknowledge support from the National Science Foundation under awards CNS-1653110 and CNS-1801479, and a Google Security & Privacy Award.

## References

[1] KeyForge and TimeForge source code. https://github.com/mspecter/KeyForge.

[2] Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest. Lightweight email signatures. In *International Conference on Security and Cryptography for Networks*, pages 288–302. Springer, 2006.

[3] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Lightweight encryption for email. In *Steps to Reducing Unwanted Traffic on the Internet Workshop, SRUTI'05*. USENIX Association, 2005.

[4] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[5] Diego F. Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. Implementing pairings at the 192-bit security level. In *International Conference on Pairing-Based Cryptography*, pages 177–195. Springer, 2012.

[6] Associated Press. DKIM verification script. https://github.com/associatedpress/verify-dkim.

[7] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC 2008*, 2008.

[8] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact E-Cash and Simulatable VRFs Revisited. In *Pairing-Based Cryptography '09*, 2009.

[9] Steven Michael Bellovin. Spamming, phishing, authentication, and privacy. 2004.

[10] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, pages 56–73, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[11] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matt Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, pages 41–55, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[12] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[13] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS - A practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2011.

[14] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 319–338.

[15] Jack Burbank, David Mills, and William Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. https://tools.ietf.org/html/rfc5905 [https://perma.cc/428T-HN3Y].

[16] John Callas, Eric Allman, Miles Libbey, Michael Thomas, Mark Delany, and Jim Fenton. DomainKeys Identified Mail (DKIM) Signatures.

[17] Jon Callas. [ietf-dkim] Thinking about DKIM and surveillance. https://mailarchive.ietf.org/arch/msg/ietf-dkim/eWKbWdYmkX_d2ki_lAbczVSj8qY [https://perma.cc/DQF6-SQNZ].

[18] Jon Callas. [ietf-dkim] DKIM Key Sizes, October 2016. http://mipassoc.org/pipermail/ietf-dkim/2016q4/017195.html [https://perma.cc/7NNX-QJUK].

[19] Jon Callas. [ietf-dkim] DKIM Key Sizes, October 2016. http://mipassoc.org/pipermail/ietf-dkim/2016q4/017207.html [https://perma.cc/K8LM-KJS7].

[20] Jan Camenisch, Rafik Chaabouni, and abhi shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 234–252, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[21] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology–CRYPTO 2004*, 2004.

[22] Charles Cazabon. getmail version 5. http://pyropus.ca/software/getmail.

[23] Don Coppersmith and Markus Jakobsson. Almost optimal hash sequence traversal. In Matt Blaze, editor, *Financial Cryptography, 6th International Conference, FC 2002, Southampton, Bermuda, March 11-14, 2002, Revised Papers*, volume 2357 of *Lecture Notes in Computer Science*, pages 102–119. Springer, 2002.

[24] D. Crocker. Internet Mail Architecture, 2009. https://tools.ietf.org/html/rfc5598.

[25] Mario Di Raimondo and Rosario Gennaro. New approaches for deniable authentication. *Journal of Cryptology*, 22(4):572–615, Oct 2009.

[26] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 409–418, New York, NY, USA, 1998. ACM.

[27] Hilda L. Fontana. Authentication Failure Reporting Using the Abuse Reporting Format. https://tools.ietf.org/html/rfc6591 [https://perma.cc/5MTF-ZD8P].

[28] Center for Strategic and International Studies (CSIS). Significant cyber incidents, 2018. https://www.csis.org/programs/cybersecurity-and-governance/technology-policy-program/other-projects-cybersecurity.

[29] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Proceedings of ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer, 2002.

[30] Google. Making email safer for you, February 2016.

[31] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[32] Lachlan J. Gunn, Ricardo Vieitez Parra, and N. Asokan. Circumventing cryptographic deniability with remote attestation, 2019.

[33] HIPAA Journal. United hospital district phishing attack impacts 2,143 patients, 2019. https://www.hipaajournal.com/united-hospital-district-phishing-attack-impacts-2143-patients/.

[34] J. Levine. RFC 8463 - A New Cryptographic Signature Method for DomainKeys Identified Mail (DKIM).

[35] Markus Jakobsson. Fractal hash sequence representation and traversal. *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT)*, pages 437–44, 2002.

[36] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. Designated verifier proofs and their applications. In *Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'96, pages 143–154, Berlin, Heidelberg, 1996. Springer-Verlag.

[37] John Klensin. RFC5321: Simple Mail Transfer Protocol, 2008.

[38] Murray Kucherawy and Elizabeth Zwicky. Domain-based Message Authentication, Reporting, and Conformance (DMARC). https://tools.ietf.org/html/rfc7489.

[39] Kurt Andersen. M3aawg DKIM Key Rotation Best Common Practices | M3aawg, March 2019. http://www.m3aawg.org/DKIMKeyRotation [https://perma.cc/4WY6-SH8K].

[40] Jeremy B. Merrill. Authenticating Email Using DKIM and ARC, or How We Analyzed the Kasowitz Emails. *ProPublica*, July 2017.

[41] Moni Naor. Deniable ring authentication. In *Advances in Cryptology — CRYPTO 2002*, pages 481–498, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[42] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2013.

[43] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5:2–13, 2002.

[44] Adrian Perrig, Dawn Song, Ran Canetti, J. D. Tygar, and Bob Briscoe. Timed efficient stream loss-tolerant authentication (TESLA): multicast source authentication transform introduction. *RFC*, 4082:1–22, 2005.

[45] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, 2001.

[46] Raphael Satter. Emails: Lawyer who met Trump Jr. tied to Russian officials. *The Associated Press*, July 2018.

[47] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Proceedings of CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[48] Michael Specter, Sunoo Park, and Matthew Green. KeyForge: Mitigating Email Breaches with Forward-Forgeable Signatures. Cryptology ePrint Archive, Report 2019/390, 2019. https://eprint.iacr.org/2019/390.

[49] Jonathan Stempel and Jim Finkle. Yahoo says all three billion accounts hacked in 2013 data theft, 2017. https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C8201.

[50] Michael Thomas. Requirements for a DomainKeys Identified Mail (DKIM) Signing Practices Protocol. https://tools.ietf.org/html/rfc5016.

[51] Nik Unger and Ian Goldberg. Deniable key exchanges for secure messaging. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1211–1223, New York, NY, USA, 2015. ACM.

[52] Nik Unger and Ian Goldberg. Improved strongly deniable authenticated key exchanges for secure messaging. *PoPETs*, 2018(1):21–66, 2018.

[53] Wikileaks. Sony Email Leak. https://wikileaks.org/sony/emails/, 2012.

[54] Wikileaks. The Global Intelligence Files: STRATFOR email leak. https://wikileaks.org/gifiles/docs/13/1328496_stratfor-.html, 2012.

[55] Wikileaks. Search the DNC Database, July 2016. https://wikileaks.org/dnc-emails/.

[56] Wikileaks. WikiLeaks: DKIM Verification, nov 2016. https://wikileaks.org/DKIM-Verification.html [https://perma.cc/H3SR-YB44].

## A Non-attributability definitions

**Definition 5** (Recipient non-attributability)**.** Email *is* non-attributable for recipients w.r.t. *$F$ if there is a PPT simulator $\mathscr{S}$ such that for any sender $S$ and recipient $R$ (with respective internal states $s, r$), for any email message $m$ and metadata $\mu$,*

$$\mathsf{Email}_s(S,R,m,\mu,t) \approx_c \mathscr{S}_r^F(S,m,\mu) \ ,$$

*where the superscript $F$ denotes black-box or query access to an interactive functionality $F$, and the subscript $r$ denotes that $\mathscr{S}$ has access to the recipient server's internal state $r$.*[22]

**Definition 6** ($\Delta$-universal non-attributability)**.** *For $\Delta \in \mathbb{N}$, an email protocol* Email *is* $\Delta$-strongly non-attributable *if there is a PPT simulator $\mathscr{S}$ such that for any sender $S$ (with internal state $s$) and recipient $R$, for any email message $m$, metadata $\mu$, and timestamp $t$, the following holds at any time $\geq t+\Delta$:*

$$\mathsf{Email}_s(S,R,m,\mu,t) \approx_c \mathscr{S}(S,R,m,\mu,t) \ .$$

Definitions 5 and 6 serve to ensure that no attacker can credibly claim to a third party[23] that he is providing her with authentic emails: the third party is in the role of distinguisher.

Note that Definition 6 is inviable if $\Delta < \hat{\Delta}$. Otherwise, the spam- and spoofing-resistance provided by DKIM would be undermined, since any outsider could use the simulator in real time to send spam email indistinguishable to the recipient from email actually sent by an honest party. Moreover, assuming the essential condition that emails are *not* universally forgeable in real time, Definition 6 implies that the behavior of any $\mathscr{S}$ must differ (distinguishably) between times $\geq t+\Delta$ and times $\leq t$. This is satisfiable only if the view of $\mathscr{S}$ changes between these time intervals: in other words, Definition 6 is satisfiable only if $\mathscr{S}$ gains some new information between these time intervals. In KeyForge and TimeForge, this additional information is made available to $\mathscr{S}$ through the public bulletin board BB or the PVTK TK, respectively. Absent some time-dependent exogenous functionality like BB or TK, Definition 6 is (straightforwardly) unsatisfiable.

[22] In fact, our constructions achieve a slightly stronger (i.e., harder to satisfy) definition where $\mathscr{S}$ cannot read $r$, but has only oracle access to signatures by $R$ (produced using key material in $r$). In practice, the latter requirement may be significantly easier to satisfy, as it is achievable by obtaining login access to an email account rather than compromising the server's secrets. However, the definition assumes direct access to $r$ for simplicity.

[23] E.g., the general public (if the allegedly stolen emails are released publicly) or a specific interested party (such as a potential buyer or disseminator of the information).

## B FFS security requirements in full

Formal details of the required properties which were omitted from Definition 1 are given below.

- *Correctness:* For all $m \in \mathscr{M}, \tau \in \mathscr{T}$, there is a negligible function $\varepsilon$ such that for all $\kappa$,

$$\Pr\left[\begin{array}{l}(vk,sk) \leftarrow \mathsf{KeyGen}(1^\kappa)\\ \sigma \leftarrow \mathsf{Sign}(sk,\tau,m)\\ b \leftarrow \mathsf{Verify}(vk,\tau,m,\sigma)\end{array} \quad : \quad b=1\right] \geq 1-\varepsilon(\kappa)\ .$$

- *Unforgeability:* For any PPT $\mathscr{A}$, there is a negligible function $\varepsilon$ such that for all $\kappa \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l}(vk,sk) \leftarrow \mathsf{KeyGen}(1^\kappa)\\ (\tau,m,\sigma) \leftarrow \mathscr{A}^{\mathsf{S}_{sk},\mathsf{E}_{sk}}(vk)\\ b \leftarrow \mathsf{Verify}(vk,\tau,m,\sigma)\\ b' = \tau \notin Q'_{\mathsf{E}} \wedge (\tau,m) \notin Q_{\mathsf{S}}\end{array} \quad : \quad b=b'=1\right] \leq \varepsilon(\kappa)\ ,$$

  where $\mathsf{S}_{sk}$ and $\mathsf{E}_{sk}$ respectively denote oracles $\mathsf{Sign}(sk,\cdot,\cdot)$ and $\mathsf{Expire}(sk,\cdot)$, $Q_{\mathsf{S}}$ and $Q_{\mathsf{E}}$ denote the sets of queries made by $\mathscr{A}$ to the respective oracles, and $Q'_{\mathsf{E}} = \bigcup_{T \in Q_{\mathsf{E}}} T$.

## C FFS vs. FSS

FSS were designed with a different goal from FFS: namely, to allow efficient key updating while preventing derivation of *past* keys from present and future keys. In contrast, our setting requires that *present and future* keys cannot be derived from past keys. The only way to achieve this property using an FFS would be to precompute a long list of secret keys and then use them *in backwards order* — this is arguably better than the simplest solution based on short-lived keys, but storing the whole list of keys is inefficient and unsatisfactory.

**Difference with forward-secure signatures** Both FFS and FSS yield a system of short-lived secret keys all corresponding to one long-lived public key. However, the definitions differ in two main ways, described below and depicted in Figure 5.

1. Forward-*secure* signatures require that past keys cannot be computed from future keys, whereas forward-*forgeable* signatures require that future keys cannot be computed from past keys.
2. Forward-*secure* signatures are designed to prevent compromise of past signatures by compromising a later secret key. All FSS secret keys are short-lived and each secret key must be derivable based solely on the previous short-lived secret key. Forward-*forgeable* signatures, in contrast, may have persistent "master secret key" material used to generate each short-lived key.

## D Further Discussion of TimeForge

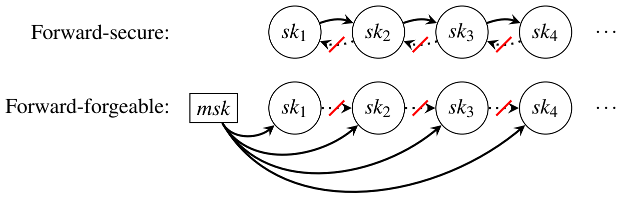

Figure 5: Forward-secure vs. forward-forgeable signatures

TimeForge can be realized using a variety of WI and ZK proof systems, combined with efficient SUF-CMA signature schemes. For example, a number of pairing-based signature schemes [7, 8, 21] admit efficient proofs of knowledge of a signature using simple Schnorr-style proofs [3]. More recent proving systems, e.g., Bulletproofs [14] and zkSNARKs (e.g., [31, 42], admit succinct proofs of statements involving arbitrary arithmetic circuits and discrete-log relationships. Using the latter schemes ensures short proofs, in the hundreds of bytes, in some cases with a small, constant verification cost. Thus, even complex PVTK proofs such as block header sequences, can potentially be reduced to a succinct TimeForge signature.

**A concrete implementation.** For our basic implementation, which signs a timestamp, we considered several proof systems. For the relatively simple proof statement used in this scheme, Bulletproofs are not appropriate for two reasons: (1) the proof sizes that result exceed 1000 bytes, and (2) these proofs do not natively support efficient signatures. zkSNARKs produce bandwidth-efficient signatures, but at a significant cost due to the need to generate a trusted setup embedding the signature verification circuit. Based on these considerations, we propose and evaluate one concrete implementation based on Schnorr-style proving techniques, made non-interactive using the Fiat-Shamir heuristic. Our approach implements TimeForge using a dedicated server that produces (weak) Boneh-Boyen signatures [10] over the current time period $t$, which is encoded as an integer in $\mathbb{Z}_q$. Let $g_1, g_2$ be generators of a pair of bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ of order $q$. Briefly, a Boneh-Boyen signature on a time period $t$ comprises a single group element $\sigma = g_1^{1/x+t}$, where $x$ represents the signing key, and the server's public key is $g_2^x$. Verification is conducted by checking the following pairing equality: $e(g_1, g_2) = e(\sigma, g_2^x g_2^t)$.

Our proposed TimeForge proof of knowledge requires the following components. First, the prover to provides a Pedersen commitment $B$ to the current time period $T_{\mathsf{current}}$ using randomness $r$. The proof also reveals the (alleged) true signing time period $T_{\mathsf{signing}}$ in cleartext (in case it is different) and attaches $\delta$. Using these values, the prover employs the homomorphic property of Pedersen commitments to derive an implicit commitment $C = g_1^{\gamma = T_{\mathsf{current}} - T_{\mathsf{signing}} - \delta} h^{r'}$, and then uses a range proof to prove that it knows a value $\gamma$ that is in the range $[1, 2^{32}]$. We use a range proof due to Camenisch, Chaabouni, and shelat [20]. Alternatively, this proof could be implemented using a Bulletproof, due to Bootle *et al.* [14].

In addition to this commitment proof and range proof, we provide two separate Schnorr-style proofs in an "OR" construction:

1. *A standard Schnorr signature on the message.* This comprises an interactive proof of knowledge of a value $sk \in \mathbb{Z}_q$ such that $PK = g^{sk}$, flattened into a signature of knowledge on the signed message, using the Fiat-Shamir heuristic. (This represents the genuine signer's signature on the message.)

2. *A proof of knowledge of a Boneh-Boyen signature on the TimeForge time period $T_{\mathsf{current}}$, signed using the TimeForge server secret key. For this construction we use a interactive zero-knowledge protocol given by Boneh, Boyen and Shacham [11, Protocol 1], flattened using the same Fiat-Shamir hash function.*