J. Cryptol. (2012) 25: 310–348
DOI: 10.1007/s00145-010-9096-4

Journal of
CRYPTOLOGY

Multi-Verifier Signatures*

Tom Roeder†
Microsoft Research, Redmond, USA
throeder@microsoft.com

Rafael Pass and Fred B. Schneider
Department of Computer Science, Cornell University, Ithaca, USA
rafael@cs.cornell.edu; fbs@cs.cornell.edu

Communicated by Ran Canetti

Received 15 August 2009
Online publication 29 January 2011

Abstract. *Multi-verifier signatures* generalize public-key signatures to a secret-key setting. Just like public-key signatures, these signatures are both transferable and secure under arbitrary (unbounded) adaptive chosen-message attacks. In contrast to public-key signature schemes, however, we exhibit practical constructions of multi-verifier signature schemes that are provably secure and are based only on pseudorandom functions in the plain model without any random oracles.

Key words. Signature schemes, Message authentication codes, Multi-verifier signatures.

1. Introduction

Public-key signatures [12,17,25] are relatively expensive to generate. Moreover, practical public-key signature schemes rely on either strong number-theoretic assumptions [5,10] or are proven secure only in the random oracle model [25]. In contrast, message authentication codes (MACs) are orders of magnitude faster and can be based on pseudorandom functions. MACs, however, rely on secret-key setup and do not provide *transferability*—the property that signed messages *accepted* by one server and forwarded to other servers will be accepted there too. Transferability is essential in many applications of digital signature schemes (e.g., in distributed systems [7,21,30]).

A natural question, then, is if the secret-key setup used for MACs can be leveraged to get more efficient, yet provably secure, digital signature schemes. We answer this question in the affirmative by introducing *multi-verifier signatures* (MVS), which generalize

* Research supported in part by AFOSR grant F9550-06-0019, National Science Foundation Grants 0430161 and CCF-0424422 (TRUST), and Microsoft Corporation.

† Work for this paper from T. Roeder performed in part while at Cornell University.

public-key signatures to a secret-key setting with a *signer* and multiple *verifiers*, each using different keys. We also provide two efficient MVS constructions:

- *Atomic Signatures* requires the signer to solve a system of linear equations. As far as we know, Atomic Signatures constitutes the first practical and provably secure signature scheme based only on symmetric-key primitives.
- *Chain Signatures* provides λ*-limited transferability*, the property that signatures can be transferred at least $\lambda - 1$ times and still be accepted by receivers. Although λ-limited transferability is weaker than transferability, it suffices in many settings (e.g., see systems [7,21,24,30], where each message is forwarded only a fixed number of times). Furthermore, for values of λ used in practical protocols, Chain Signatures outperforms the fastest implementations of public-key signature schemes (even though these public-key signature schemes are only secure in the random oracle model [25]).

Atomic Signatures and Chain Signatures are based only on the existence of pseudorandom functions and do not assume random oracles. One-way functions (hence pseudorandom functions) are known [22,26] to be sufficient to construct public-key signature schemes, but existing signature constructions based on one-way functions are too inefficient to be used in practice.

Our MVS constructions require an unusual secret-key setup—pairwise shared keys distributed in such a way that the signer does not know which key corresponds to which verifier. This prevents the signer from creating signatures that would be accepted by some verifiers but not others. The required secret-key setup is easily implemented, for example, in an operating system (OS) or small distributed service. Processes in an OS already trust the OS, so the OS can distribute shared keys when a process is created. Similarly, a small distributed service (e.g., [7,21,30]) that is managed by a single administrator can distribute keys before the service begins executing. In these practical settings, MVS schemes provide a speed advantage over common public-key signature schemes.

The idea of constructing signature schemes in a secret-key setting began with Chaum and Roijakkers [9]. However, none of that work or its successors (e.g., [18,19,24,27, 29]) satisfies fully *adaptive security*—that unforgeability holds even with respect to an adversary that can see an a priori unbounded number of signatures on messages of its choice. In modern network environments, supporting adaptive security is important.

Although our constructions are relatively clean and simple, the proofs of security turn out to be significantly more complicated. A major theme in both is to seemingly weaken the construction in order to gain adaptive security. This idea goes back to the zero-knowledge argument construction of Feige and Shamir [15], where a trapdoor is embedded into the proof systems in order to facilitate zero-knowledge simulation. Our schemes rely on a similar argument. However, in contrast to previous applications of this technique, we can show in our scenario that this weakening is essential. In fact, for the case of Chain Signatures, the most natural implementation using MACs does not yield a scheme secure under adaptive attacks (see Remark 4.1 in Sect. 4). Yet, surprisingly, if we slightly weaken the scheme by making public to whom a certain subset of the keys belongs, then the scheme is provably secure.

The paper proceeds as follows. Assumptions and a game-based definition of MVS schemes are discussed in Sect. 2. In Sect. 3 and Sect. 4, we present constructions of our

MVS schemes and prove that they work. Finally, Sect. 5 gives performance results from an implementation of our schemes, and related work is discussed in Sect. 6.

2. Definitions of Multi-Verifier Signature Schemes

Our constructions of MVS schemes rely on MACs, which take a message m and a key k as input and output a *tag* that can be used to authenticate m given k. The traditional model of MAC security [2] requires that it be hard for an adversary $\mathcal{A}$ to generate a message m and tag τ that will be accepted by a receiver, even if $\mathcal{A}$ has access to a MAC oracle (as long as $\mathcal{A}$ has not already requested that m be signed by the oracle). We write $\mathcal{A}_r$ for an adversary $\mathcal{A}$ using randomness r.

We require the MAC to satisfy a stronger property (similar to Bellare, Goldreich, and Mityagin [3]), called *Chosen Tag Attack* (*CTA*) *Unforgeability*, which additionally gives adversaries access to a verification oracle $\mathsf{VF}(m, \tau, k)$ such that

$$\mathsf{VF}(m, \tau, k) = \begin{cases} 1 & \text{if } \mathsf{MAC}(m, k) = \tau, \\ 0 & \text{otherwise.} \end{cases}$$

If $\mathsf{VF}(m, \tau, k) = 1$ holds, then the tag is accepted.[1]

Let $\mathsf{MACreq}(\mathcal{A}, m, k, r)$ be a predicate that is true if a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}_r$ requested m from its MAC oracle using key k in a given execution. Then, for a given security parameter d, we can define the following property.

CTA Unforgeability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that

$$\begin{aligned} \Pr\big[k \leftarrow \mathsf{Gen}\big(1^d\big); r \leftarrow \{0,1\}^\infty; \\ m, \tau \leftarrow \mathcal{A}_r^{\mathsf{MAC}(\cdot,k),\mathsf{VF}(\cdot,\cdot,k)}\big(1^d\big) : \\ \neg\mathsf{MACreq}(\mathcal{A}, m, k, r) \wedge \mathsf{VF}(m, \tau, k) = 1\big] \leq \epsilon(d). \end{aligned}$$

Pseudorandom functions (see [16] for an overview) can be used to construct MACs satisfying CTA Unforgeability.

2.1. *Multi-Verifier Signatures*

We define an MVS scheme to be a triple (Gen, Sign, Ver) of algorithms that depend on a set I of n *verifiers*; each verifier may use a different key when calling Ver. And in an implementation of an MVS scheme, each server would execute a verifier with a different key. These algorithms operate as follows, where argument d is a security parameter, and b specifies a given number of bits.

[1] It might seem like an adversary given a MAC oracle can always simulate the execution of the verification oracle for a deterministic MAC by calling the MAC oracle and checking if its output matches the given tag. However, in some security definitions, the adversary is not allowed to call the MAC oracle on all messages, so this simulation will not always be possible. In these cases, the verification oracle gives the adversary more power.

- $\mathsf{Gen}(1^d, 1^n)$ is a PPT algorithm that outputs a vector $\boldsymbol{k}$ of keys for the signer and a vector $\boldsymbol{K}_j$ of keys for each verifier $j \in I$. Key k in $\boldsymbol{k}$ or $\boldsymbol{K}_j$ (for any $j \in I$) is an element of $\{0,1\}^b$.
- $\mathsf{Sign}(m, \boldsymbol{k})$ takes $m \in \{0,1\}^b$ and produces[2] a tag τ.
- For each $j \in I$, $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ produces a value that is either ∞, 0, an element of $\mathbb{N}_{>0}$, or $\perp$.[3] If $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ returns ∞, then we say that τ is *accepted* by j, and if $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ returns 0, then we say that τ is not accepted by j. A return value $\lambda \in \mathbb{N}_{>0}$ means that τ is accepted by j, and also that there is a lower bound $\lambda - 1$ on the number of times this tag can be transferred and still cause verifiers at other correct receivers to return a value in $\mathbb{N}_{>0}$. If $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$, then j sets a state variable v_j to $\perp$; verifier j thereafter believes the signer to be compromised.

Note that the above description implies that Ver is a stateful algorithm: whenever Ver produces $\perp$ for a message purporting to come from signer i, it decides that i is compromised and remembers this fact. Thereafter, Ver will only return $\perp$ for message and tag pairs purporting to come from i.

Let λ be an element of $\mathbb{N}^\infty \triangleq \mathbb{N}_{>0} \cup \{\infty\}$, and let $\mathsf{req}(\mathcal{A}, m, \boldsymbol{k}, r)$ be a predicate that is true if an adversary $\mathcal{A}_r$ requested m from its signing oracle using keys $\boldsymbol{k}$ in a given execution. MVS schemes satisfy four properties: λ-Completeness, Unforgeability, Non-Accusability, and Transferability.[4]

λ-Completeness stipulates that, for any $\boldsymbol{K}_j$ and any message m, a tag τ generated by $\mathsf{Sign}(m, \boldsymbol{k})$ must cause $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ to return a value that is greater than or equal to λ. Note that ∞-Completeness thus requires $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ to return ∞; the value ∞ means that a message and tag can be transferred an arbitrary number of times, just like signed messages in public-key signatures. Formally, we can state this property as follows.

λ-Completeness. There exists a negligible function ϵ such that for any message m and any $j \in I$,

$$\Pr\left[\left(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i \in I}\right) \leftarrow \mathsf{Gen}\left(1^d, 1^n\right) : \mathsf{Ver}\left(m, \mathsf{Sign}(m, \boldsymbol{k}), \boldsymbol{K}_j\right) < \lambda\right] \leq \epsilon(d, n).$$

As with public-key signature schemes, Unforgeability requires that no adversary $\mathcal{A}$ be able to generate a message m and tag τ that will be accepted by any correct verifier if $\mathcal{A}$ has not previously requested a tag for m from its signing oracle, even if $\mathcal{A}$ might have compromised any set I' of verifiers. Formally, we can state this property as follows.

[2] For simplicity of exposition, we define MVS schemes for b-bit messages only. These definitions are easily extended to arbitrary-length messages. Our constructions are described for both fixed-length and arbitrary-length messages.

[3] Assume that ∞ satisfies only the following properties: $\infty - \infty = 0$, $\forall a \in \mathbb{N} \ : \ \infty > a$, and $\forall a \in \mathbb{N} \ : \ |\infty - a| = |a - \infty| = \infty$.

[4] Each of these properties involves a parameter ϵ which represents a bound on the probability of the property being violated. When necessary, we will refer to this bound as a parameter of the property (e.g., writing "Transferability with parameter ϵ").

Unforgeability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big); r \leftarrow \{0,1\}^\infty; \\ (m,\tau) \leftarrow \mathcal{A}_r^{\mathsf{Sign}(\cdot,\boldsymbol{k}),\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \{\boldsymbol{K}_i\}_{i\in I'}\big) : \\ \neg\mathsf{req}(\mathcal{A}, m, \boldsymbol{k}, r) \wedge \big(\exists j \in I - I' \,:\, \mathsf{Ver}(m,\tau,\boldsymbol{K}_j) \in \mathbb{N}^\infty\big)\Big] \leq \epsilon(d,n).$$

Non-Accusability requires that no adversary $\mathcal{A}$ be able to generate a message and tag pair that causes any correct verifier to return $\perp$ if the signer is not compromised, even if $\mathcal{A}$ can request signatures on arbitrary messages and has compromised an arbitrary subset of verifiers. Formally, we can state this property as follows.

Non-Accusability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big); \\ (m,\tau) \leftarrow \mathcal{A}^{\mathsf{Sign}(\cdot,\boldsymbol{k}),\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \{\boldsymbol{K}_i\}_{i\in I'}\big) : \\ \big(\exists j \in I - I' \,:\, \mathsf{Ver}(m,\tau,\boldsymbol{K}_j) = \perp\big)\Big] \leq \epsilon(d,n).$$

Transferability requires that even a compromised signer is unable to create a tag on which any pair of correct verifiers return values that differ by more than 1; this property implies that if verification of a message m and tag τ at any correct verifier returns v, then m and τ can be transferred between correct verifiers at least $v-1$ times. A tag that does not satisfy Transferability is said to be *split*.[5]

Our notion of Transferability is slightly different than the notion of Transferability guaranteed by public-key signatures. In particular, a verifier j that returns a value in $\mathbb{N}^\infty$ for a message m and tag τ is not guaranteed that any noncompromised verifier j' to which it passes m and τ will also return a value in $\mathbb{N}^\infty$; the other possibility is that j' returns $\perp$—which means that j' has acquired proof that the signer is compromised.

Formally, we can state this property as follows.

Transferability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big); \\ (m,\tau) \leftarrow \mathcal{A}^{\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I'}\big) : \\ \big(\exists j, j' \in I - I' \,:\, \mathsf{Ver}(m,\tau,\boldsymbol{K}_j) \neq \perp \wedge \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'}) \neq \perp \\ \wedge \big|\mathsf{Ver}(m,\tau,\boldsymbol{K}_j) - \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'})\big| > 1\big)\Big] \leq \epsilon(d,n).$$

[5] Verification algorithms for MVS schemes allow some verifiers to return $\perp$ and conclude nothing about the tag. Each tag divides the correct verifiers into two groups: those that accept the tag and those that find the signer compromised. This specification is reminiscent of Crusader's Agreement [13], where the correct receivers are divided into those that agree on a single value and those that know the sender to be compromised.

It might seem like Transferability is unnecessarily complex and that tags generated to satisfy $\lambda = 2$ are always sufficient, since then the sender would find $\lambda = 2$, and all other verifiers would find $\lambda = 1$, so the tag would be accepted at all verifiers. However, consider a signer i that creates a message that must be sent to verifier j and then sent by j to another verifier j'. Signer i creates a tag that satisfies $\lambda = 2$, so i knows that j will find the tag to satisfy at least $\lambda = 1$. But i does not know if j will be willing to forward the message to j', since j's value of $\lambda = 1$ would lead it to believe that j' could find $\lambda = 0$, hence that j' might not accept the tag. So, the value of λ must be chosen to be at least the number of times the signature is to be transferred.[6]

An MVS scheme that satisfies λ-Completeness for $\lambda \in \mathbb{N}_{>0}$ is called a λ-*MVS* scheme; tags generated by this scheme can be transferred at least $\lambda - 1$ times. An MVS scheme that satisfies ∞-Completeness is called an ∞-*MVS* scheme; tags generated by this scheme can be transferred an unlimited number of times. A stronger version of Transferability, called *Perfect Transferability*, would require that $\epsilon(d, n) = 0$ in the definition of Transferability; Appendix C shows that ∞-MVS schemes satisfying Perfect Transferability are essentially public-key signature schemes.[7]

Similar to public-key signature schemes, an MVS scheme can satisfy *Strong Unforgeability*, which requires that if $\mathcal{A}$ did not receive m and τ from its signing oracle, then m and τ will not cause any verifier to return a value greater than 0 (leading this verifier to accept the pair) or return $\perp$ (leading this verifier to conclude that the signer is compromised). Define predicate $\mathsf{recv}(\mathcal{A}, (m, \tau), \boldsymbol{k}, r)$ to be true if $\mathcal{A}$, using randomness r, received m and τ from the signing oracle using keys $\boldsymbol{k}$. Formally, the property is written as follows:

Strong Unforgeability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big); r \leftarrow \{0,1\}^\infty;$$
$$(m, \tau) \leftarrow \mathcal{A}_r^{\mathsf{Sign}(\cdot,\boldsymbol{k}),\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \{\boldsymbol{K}_i\}_{i\in I'}\big) :$$
$$\neg\mathsf{recv}(\mathcal{A}, (m, \tau), \boldsymbol{k}, r) \wedge \big(\exists j \in I - I' \ : \ \mathsf{Ver}(m, \tau, \boldsymbol{K}_j) \neq 0\big)\Big] \leq \epsilon(d, n).$$

Note that Strong Unforgeability implies both Unforgeability and Non-Accusability. An MVS scheme that satisfies Strong Unforgeability in addition to λ-Completeness and Transferability is called a *Strong* λ-*MVS* scheme.

3. Atomic Signatures

Atomic Signatures is a Strong ∞-MVS scheme in which a signer computes a tag for a message m by solving a system of linear equations generated over MACs of m with the verifiers' keys. Unlike MACs and public-key signature schemes, generation algorithm

[6] This example demonstrates that our definition of Transferability is closely connected to the idea of knowledge in knowledge logic [14]; to send the message and tag in the example, the signer must know that the first verifier will know that the second verifier will accept the tag. And this requires a value of $\lambda = 3$.

[7] Boneh, Durfee, and Franklin [4] show a related lower bound: public-key signatures are required for short collusion-resistant multicast MAC constructions.

$\mathsf{Gen}^{\mathrm{AS}}$ for Atomic Signatures distributes disjoint, equal-sized sets of random keys to each verifier; a verifier shares each key in each set with the signer, but the signer does not know which keys it shares with which verifier.

More precisely, for n verifiers and security parameter d, generation algorithm $\mathsf{Gen}^{\mathrm{AS}}(1^d, 1^n)$ for Atomic Signatures produces dn keys $k_1, k_2, \ldots, k_{dn}$, where, for each j such that $1 \leq j \leq dn$, $k_j \in \{0,1\}^b$ is a key for a MAC. $\mathsf{Gen}^{\mathrm{AS}}$ also generates dn random vectors $\boldsymbol{z}_j = \langle z_{j,1}, z_{j,2}, \ldots, z_{j,dn} \rangle$ (one vector for each key k_j), where for each j and i such that $1 \leq j \leq dn$ and $1 \leq i \leq n$, each entry $z_{j,i}$ is an element of $\{0,1\}^b$. $\mathsf{Gen}^{\mathrm{AS}}$ then sets $\boldsymbol{k}$ to be a vector of key pairs $\langle (k_1, \boldsymbol{z}_1), (k_2, \boldsymbol{z}_2), \ldots, (k_{dn}, \boldsymbol{z}_{dn}) \rangle$. And $\mathsf{Gen}^{\mathrm{AS}}$ creates n vectors $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_n$ that each contain a unique, randomly chosen set of d key pairs from $\boldsymbol{k}$ such that $\boldsymbol{K}_i$ and $\boldsymbol{K}_j$ are disjoint for each distinct pair i and j of indices. Set $\boldsymbol{k}$ is called the *signing key pairs*, and the n vectors $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_n$ are called the *verifying key pairs*. We say that a verifier j *owns* each key pair $(k, \boldsymbol{z})$ in $\boldsymbol{K}_j$.

Signing algorithm $\mathsf{Sign}^{\mathrm{AS}}(m, \boldsymbol{k})$ takes message m and signing key pairs $\boldsymbol{k}$ as input[8] and outputs a vector $\boldsymbol{A}^d(m) = A_1, A_2, \ldots, A_{dn}$ of subtags, obtained by solving the following equation, where b-bit strings are treated as elements in the finite field GF(2^b):

$$\begin{pmatrix} \mathsf{MAC}(m, k_1) \\ \mathsf{MAC}(m, k_2) \\ \vdots \\ \mathsf{MAC}(m, k_{dn}) \end{pmatrix} = \begin{pmatrix} z_{1,1} & z_{1,2} & \cdots & z_{1,dn} \\ z_{2,1} & z_{2,2} & \cdots & z_{2,dn} \\ \vdots & \vdots & \vdots & \vdots \\ z_{dn,1} & z_{dn,2} & \cdots & z_{dn,dn} \end{pmatrix} \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_{dn} \end{pmatrix}. \tag{1}$$

Recall that solving (1) corresponds to solving the dn instances ($1 \leq j \leq dn$) of the following equation simultaneously:

$$\mathsf{MAC}(m, k_j) = \sum_{t=1}^{dn} z_{j,t} A_t. \tag{2}$$

Define predicate $\mathsf{roweq}(m, \boldsymbol{A}, (k_j, \boldsymbol{z}_j))$ to be true exactly when (2) holds for m, $\boldsymbol{A}$, and $(k_j, \boldsymbol{z}_j)$.

Verification algorithm $\mathsf{Ver}^{\mathrm{AS}}(\cdot, \cdot, \boldsymbol{K}_j)$ keeps state v_j to record that it has been called with a message and tag that indicate the signer is compromised. Verification on a message m and tag τ for verifier j operates as follows:

- If v_j is set to $\perp$, then $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$.
- Otherwise, if $\mathsf{roweq}(m, \tau, (k, \boldsymbol{z}))$ holds for all d pairs $(k, \boldsymbol{z})$ in $\boldsymbol{K}_j$, then $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ returns ∞.
- If no $\mathsf{roweq}(m, \tau, (k, \boldsymbol{z}))$ holds for any $(k, \boldsymbol{z})$ in $\boldsymbol{K}_j$, then $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ returns 0.
- Otherwise, there is some key pair $(k, \boldsymbol{z}) \in \boldsymbol{K}_j$ for which $\mathsf{roweq}(m, \tau, (k, \boldsymbol{z}))$ holds, and a different key pair $(k', \boldsymbol{z}') \in \boldsymbol{K}_j$ for which $\mathsf{roweq}(m, \tau, (k', \boldsymbol{z}'))$ does not hold. So, $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$ and sets v_j to $\perp$.

[8] Atomic Signatures as described can sign arbitrary-length messages as long as the MAC can generate tags for arbitrary-length messages.

Remark 3.1. This verification scheme is based on the idea that it is hard for an adversary that does not know a key (k, z) in $\boldsymbol{k}$ to produce a message m and tag τ that will cause $\mathsf{roweq}(m, \tau, (k, z))$ to hold. The intuition behind the proof is that the MAC is hard to forge.

This scheme also relies on it being hard for an adversary that has not compromised the signer to produce a message and tag pair that causes any noncompromised verifier to return $\perp$. This property must hold even if the adversary can see a polynomial number of messages and tags for messages of its choice. The intuition behind the proof of this property is that it is hard to find new solutions to underdetermined equations over variables with randomly-chosen, unknown coefficients (the pseudorandom MAC values and the random rows of the matrix).

However, it is easy for a compromised signer to cause one verifier to return $\perp$: the signer can create a new set of signing keys $\boldsymbol{k}'$ by replacing one of the keys in $\boldsymbol{k}$ with a random bit string. Then the signer uses $\boldsymbol{k}'$ to generate and solve (1). Some verifier j will then only find $d - 1$ satisfied instances of $\mathsf{roweq}(m, \tau, (k, z))$ and will return $\perp$. The only task that must be difficult for a compromised signer is creating a message and tag pair that cause one noncompromised verifier to return ∞ and another to return 0.

Remark 3.2. The verification and signing algorithms for Atomic Signatures as described above are expensive to compute: each verifier has $d + d^2n$ keys, and the signer must solve a matrix equation at cost $O(d^3n^3)$ to generate a signature. Both of these costs can be reduced.

One way to reduce the number of keys is for each verifier j to share $2d$ keys with the signer instead of $d + d^2n$—instead of keys (k, z), a verifier shares a pair (k^1, k^2), where keys k^1 and k^2 are elements of $\{0, 1\}^b$. Key k^1 is used to compute MACs in the place of k in (1), and k^2 is used to generate matrix row elements $z_t = \mathsf{MAC}(t, k^2)$ for t in $\{1, 2, \ldots, dn\}$; each verifier then only needs $2d$ keys, two for each of its d rows. The proofs follow in the same way as with a randomly chosen matrix but with an extra hybrid step to go from random matrix elements to matrix elements generated by a pseudorandom function.[9]

The cost of generating a signature can be reduced further. $\mathsf{Gen}^{\mathrm{AS}}$ can produce a prefactored matrix (using the LU factorization, for instance) for use by $\mathsf{Sign}^{\mathrm{AS}}$ on the right-hand side of (1). Factoring is cost-effective here, because the matrix is independent of the message to be signed—factoring costs $O(d^3n^3)$ but only needs to be done once, and solutions to (1) can be found for a factored matrix in time $O(d^2n^2)$.

3.1. *Properties of Atomic Signatures*

Since a Strong ∞-MVS scheme must satisfy ∞-Completeness, Strong Unforgeability, and Transferability, we prove that Atomic Signatures is a Strong ∞-MVS by proving two lemmas. The first establishes that Atomic Signatures satisfies ∞-Completeness and Strong Unforgeability; the second that it satisfies Transferability.

Lemma 1. *If the* MAC *is a pseudorandom function, then Atomic Signatures satisfies ∞-Completeness and Strong Unforgeability.*

[9] The adversary in this case is given oracle access to the MAC functions, so it can compute signatures efficiently.

Proof. **∞-Completeness.** This follows from nonsingularity of the matrix in (1), since a nonsingular matrix allows a solution $\boldsymbol{A}^d(m)$ to be found for (1). Any solution satisfies ∞-Completeness by definition, since $\mathsf{roweq}(m, \boldsymbol{A}^d(m), (k_j, z_j))$ will hold for each row j. The probability of a random matrix of size $dn \times dn$ over a finite field of size 2^b being singular is known [8] to be $1 - \prod_{i=1}^{dn}(1 - 1/2^{bi})$, which is negligible when d and n are polynomial in b.

Strong Unforgeability. Lemmas 10–12 from Appendix A simplify the problem to the case where all but one verifier q are compromised, adversaries are allowed no verification oracle queries, and, for all indices j representing rows for verifier q, $\mathsf{MAC}(\cdot, k)$ for row j is replaced by a random function $v_j(\cdot)$.

For such an adversary to violate Strong Unforgeability, it must produce a message m and tag τ' such that it has never received (m, τ') from the signing oracle, and $\mathsf{Ver}^{\mathrm{AS}}(m, \tau', \boldsymbol{K}_q) \neq 0$ holds; this occurs exactly when $\mathsf{roweq}(m, \tau', (k, z))$ holds for at least one pair (k, z) in $\boldsymbol{K}_q$. We consider two cases. In Case 1, m has been received from the signing oracle, but with a different tag τ. In Case 2, m has never been received from the signing oracle. Since the MAC for a given row i is replaced with a random function v_i, we write (v_i, z_i) in the place of key information (k_i, z_i).

Case 1. Consider an adversary that makes a series of signing oracle queries $m_1, m_2, \ldots, m_p$, receives responses $\tau_1, \tau_2, \ldots, \tau_p$ and finally outputs a message m_w (where $1 \leq w \leq p$) and tag τ' such that $\tau' \neq \tau_w$. Note that τ' must not equal τ_w because $\mathcal{A}$ cannot return a message and tag pair that it received from its signing oracle. We show that for any choices of $m_1, m_2, \ldots, m_p$, $\tau_1, \tau_2, \ldots, \tau_p$, the probability that τ' is a valid tag for m_w is negligible. In other words, we sample the space of keys and random functions, and, for any choice of τ' by the adversary (who is given $\tau_1, \tau_2, \ldots, \tau_p$), we compute the probability of τ' violating Strong Unforgeability conditional on the fact that $\tau_1, \tau_2, \ldots, \tau_p$ are correct responses from the signing oracle.

We consider the case where the adversary violates Strong Unforgeability for a particular message m_w and produces a tag τ' satisfying the equation for m_w for a particular row i. We will then use the Union Bound to bound the probability for any row and any message. Let (v, z) be the key information for row i. Recall that message m and tag τ induce an equation for key information (v, z): $v(m) = \sum_{t=1}^{dn} z_t \tau_t$.

The probability that a tag τ' produced by an adversary satisfies the given equation is bounded by the following conditional probability for any choice of $\tau', \tau_1, \tau_2, \ldots, \tau_p$ where $\tau' \neq \tau_w$:

$$\Pr_{v,z}\left[v(m_w) = \sum_{t=1}^{dn} z_t \tau'_t \;\middle|\; \forall j \in \{1, \ldots, p\} \;:\; v(m_j) = \sum_{t=1}^{dn} z_t \tau_{j,t} \right].$$

This is the probability that a given tag τ' violates Strong Unforgeability using a given row with randomly chosen coefficients, constrained by the fact that τ_1 through τ_p satisfy the equation for this row. And the definition of conditional probability states that, for events A and B, $\Pr[A \mid B] = \Pr[A \wedge B] / \Pr[B]$. For the conditional probability above, event A is $v(m_w) = \sum_{t=1}^{dn} z_t \tau'_t$, and event B is $\forall j \in \{1, \ldots, p\} \;:\; v(m_j) = \sum_{t=1}^{dn} z_t \tau_{j,t}$. So, the probability can be split into two components: (i) $\Pr_{v,z}[\forall j \in \{1, \ldots, p\} \;:\; v(m_j) = \sum_{t=1}^{dn} z_t \tau_{j,t}]$ and (ii) $\Pr_{v,z}[v(m_w) =$

$\sum_{t=1}^{dn} z_t \tau'_t \ \wedge \ \forall j \in \{1, \ldots, p\} \ : \ v(m_j) = \sum_{t=1}^{dn} z_t \tau_{j,t}]$. We will consider these two components separately. For each component, we can consider the values of τ' and $\tau_1, \tau_2, \ldots, \tau_p$ as fixed and the (v, z) as variables.

Fixing any values for the dn variables $z_1, z_2, \ldots, z_{dn}$ in component (i) induces a single solution (namely $v(m_i) = \sum_{t=1}^{dn} z_t \tau_{i,t}$) for the values of $v(m_i)$. This means that there are 2^{bdn} ways to choose the variables to satisfy the equations. And there are, a priori, $2^{b(dn+p)}$ ways to choose the values for $dn + p$ variables. Since the probability is over the random choice of variables, this probability can be computed as the quotient $2^{bdn}/2^{b(dn+p)} = 1/2^{-bp}$.

In component (ii), each way of setting $dn - 1$ of the variables leads to a unique solution for the remaining $p + 1$ variables. To see why, recall that vectors τ' and τ_w must differ. So, they must differ in at least one of their dn entries; suppose, without loss of generality, that they differ in position a. If we set the $dn - 1$ variables $z_1, z_2, \ldots, z_{a-1}, z_{a+1}, \ldots, z_{dn}$ to any value, then we are left with p equations of the form $v(m_i) = \tau_{i,a} z_a + c_i$ for $i \in \{1, \ldots, p\}$ and constants c_i and one equation $v(m_w) = \tau'_a z_a + c'$ for some constant c'. Subtracting the equation $v(m_w) = \tau_{w,a} z_a + c_w$ from $v(m_w) = \tau'_a z_a + c'$ eliminates $v(m_w)$ and leaves an equation that uniquely determines the value of z_a, since $\tau'_a - \tau_{w,a} \neq 0$. This value for z_a then uniquely determines the values of $v(m_1), v(m_2), \ldots, v(m_p)$, by definition. So, there are $2^{b(dn-1)}$ solutions in total. And, as before, there are $2^{b(dn+p)}$ ways to choose values for these variables. So, the probability is $2^{b(dn-1)}/2^{b(dn+p)} = 2^{-b(p+1)}$.

We can use the definition of conditional probability and the two component probabilities to compute the bound on an adversary violating Strong Unforgeability using a particular equation and a particular message: $2^{-b(p+1)}/2^{-bp} = 2^{-b}$. The Union Bound over the d equations and p possible messages then gives a general bound of $pd/2^b$, which is negligible.

Case 2. Now, suppose that m was never received from the signing oracle, and consider any pair m and τ generated by adversary $\mathcal{A}$. Since $\mathcal{A}$ never received m from the signing oracle, no function of the values $v_i(m)$ for row i has been seen by $\mathcal{A}$. So, the output of $\mathcal{A}$ is independent of $v_i(m)$. Fix a key (v_j, z_j) in the keys owned by verifier q and suppose, without loss of generality, that the z_j are all known to the adversary. For a given m, there is one choice of the value of $v_j(m)$ such that $v_j(m) = \sum_{t=1}^{dn} z_{j,t} \tau_t$ holds, and there are a total of 2^b ways to choose the value of $v_j(m)$.

Since the choice of τ is independent of $v_j(m)$, and v_j is a random function, the probability of τ satisfying the given row equation is $1/2^b$. The Union Bound then gives the probability of (m, τ) violating Strong Unforgeability for any of the d row equations to be $d/2^b$, which is negligible.

Then, Lemmas 10–12 give a polynomial increase in each bound to get back to the general case. But a polynomial increase of a negligible function still leaves it negligible. So, Atomic Signatures satisfies Strong Unforgeability. □

Proving that Atomic Signatures satisfies Transferability is more challenging. A compromised signer able to generate a split tag must have knowledge about which verifier owns which keys. So, we devise a game over the signing keys, called *Idealized Random Keys*, by which we show that no adversary gets enough information to divide the signing

keys into two disjoint sets, where each set contains all the keys owned by some verifier; we call this property *Non-Separability*. Then, given an adversary that can violate Transferability, we produce a new adversary that can divide the signing keys into two such disjoint sets. This shows that Non-Separability of Idealized Random Keys reduces to Transferability of Atomic Signatures. And we also show that Idealized Random Keys satisfies Non-Separability, which means that Atomic Signatures satisfies Transferability.

3.2. *Idealized Random Keys*

Idealized Random Keys is a game between a *requester* and a set I of n *checkers* that each own a set of keys. The requester can only perform *ownership queries*: asking checkers about the ownership of keys. Checkers return $\perp$ in response to ownership queries about keys they own. The important property of Idealized Random Keys, called *Non-Separability*, is that no adversary $\mathcal{A}$ can use ownership queries to separate the set of keys into two disjoint subsets, each of which contains all the keys owned by some checker that has not already returned $\perp$.

Formally, Idealized Random Keys consists of a pair $(\mathsf{Gen}^{\mathrm{IR}}, \mathsf{Check}^{\mathrm{IR}})$ of algorithms that operate as follows:

- $\mathsf{Gen}^{\mathrm{IR}}(1^d, 1^n)$ generates a vector $\boldsymbol{k}$ of dn keys uniformly at random and partitions them into n disjoint sets $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_n$ of size d, each set owned by one checker $j \in I$.
- Check algorithm $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ keeps state s_j to record whether the requester has ever made an ownership query to j for an element of $\boldsymbol{K}_j$. When requester i makes an ownership query on a given key $k \in \boldsymbol{k}$, $\mathsf{Check}^{\mathrm{IR}}(k, \boldsymbol{K}_j)$ operates as follows.
  - if $k \in \boldsymbol{K}_j$ or $s_j = \perp$, then $\mathsf{Check}^{\mathrm{IR}}(k, \boldsymbol{K}_j)$ returns $\perp$ and sets s_j to $\perp$.
  - Otherwise, $\mathsf{Check}^{\mathrm{IR}}(k, \boldsymbol{K}_j)$ returns 1.

This behavior corresponds to information adversaries can glean from attacks on Atomic Signatures. For instance, a compromised signer j can create a tag τ for any message m by solving (1) using correct keys for all rows but one. Then j sends m and τ to a verifier j'. If j' owns the keys for this row, then j' will return $\perp$. This corresponds to a requester asking checker j' about a key in $\boldsymbol{K}_{j'}$. But if j' does not own the keys for this row, then j' will return ∞; so, j learns that j' does not own the keys for this row. This corresponds to a requester asking checker j' about a key it does not own.

Non-Separability can be written formally as follows:

Non-Separability. For all adversaries $\mathcal{A}$, there is a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\begin{aligned}
&\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}^{\mathrm{IR}}\big(1^d, 1^n\big);\\
&\quad (K, K') \leftarrow \mathcal{A}^{\{\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I'}\big):\\
&\quad \big(\exists j, j' \in I - I' \;:\; s_j \neq \perp \wedge s_{j'} \neq \perp\\
&\qquad\qquad \wedge \boldsymbol{K}_j \subseteq K \wedge \boldsymbol{K}_{j'} \subseteq K' \wedge K \cap K' = \emptyset\big)\Big] \leq \epsilon(d, n).
\end{aligned}$$

To show that Non-Separability holds for Idealized Random Keys for a given negligible parameter ϵ_0, we first consider the case $n = 2$, then provide a reduction from $n = 2$

to general n. These proofs will show how to choose d for a given n so that Idealized Random Keys satisfies Non-Separability for any $\epsilon_0 > 0$,

Lemma 2. *Idealized Random Keys for* 2 *checkers satisfies Non-Separability with parameter* $\epsilon = \binom{2d}{d}^{-1}$.

Proof. Consider a passive adversary $\mathcal{B}$ that never makes any ownership queries to its $\mathsf{Check}^{\mathrm{IR}}$ oracles but produces sets K and K' that violate Non-Separability for some value of parameter ϵ. Keys are distributed randomly, so $\mathcal{B}$ has no information about which keys correspond to which checkers. Therefore, $\mathcal{B}$'s output is independent of the distribution of keys to checkers. There are $\binom{2d}{d}$ choices for K and K', only one of which violates Non-Separability. So, $\mathcal{B}$'s probability of outputting this K and K' is $\epsilon = \binom{2d}{d}^{-1}$.

Now suppose that some adversary $\mathcal{A}$, potentially making ownership queries to its $\mathsf{Check}^{\mathrm{IR}}$ oracles, violates Non-Separability with some parameter ϵ_0. When $\mathcal{A}$ succeeds, no ownership queries to its oracles can have returned $\perp$, since there are only two checkers, j and j', and neither s_j nor $s_{j'}$ can be $\perp$ for Non-Separability to be violated. So, $\mathcal{A}$'s ownership queries must always have returned 1. This means that $\mathcal{B}$ can run $\mathcal{A}$ and simulate these ownership queries by always returning 1. $\mathcal{B}$ then returns whatever $\mathcal{A}$ returns. Whenever $\mathcal{B}$'s simulation would be incorrect (because $\mathsf{Check}^{\mathrm{IR}}$ should have returned $\perp$), $\mathcal{A}$ would have received $\perp$ and failed to violate Non-Separability. So, $\mathcal{B}$ succeeds at least as often as $\mathcal{A}$. Then, $\mathcal{A}$'s success probability ϵ_0 is also no better than $\binom{2d}{d}^{-1}$, since this is an upper bound on any passive adversary $\mathcal{B}$. □

Theorem 3. *Idealized Random Keys for n checkers satisfies Non-Separability with parameter* $\epsilon = \binom{n}{2} / \binom{2d}{d}$.

Proof. We proceed by contradiction. Suppose adversary $\mathcal{A}$ violates Non-Separability for n checkers for some $I' \subseteq I$ with parameter $\epsilon > \binom{n}{2} / \binom{2d}{d}$. We construct $\mathcal{B}$ that violates Non-Separability for Idealized Random Keys with 2 checkers with parameter $\epsilon' > \binom{2d}{d}^{-1}$, which contradicts Lemma 2.

$\mathcal{B}$ is given $\boldsymbol{k}$ as well as $\mathsf{Check}^{\mathrm{IR}}$ oracles for its two checkers and proceeds to construct keys for $\mathcal{A}$ and simulate $\mathcal{A}$'s oracles:

1. $\mathcal{B}$ calls $\mathsf{Gen}^{\mathrm{IR}}(1^d, 1^n)$ to get $\boldsymbol{k}'$ and $\{\boldsymbol{K}_i\}_{i \in I}$.
2. $\mathcal{B}$ then chooses j and j' uniformly at random from $I - I'$ and forms $\boldsymbol{k}'' = (\boldsymbol{k}' - (\boldsymbol{K}_j \cup \boldsymbol{K}_{j'})) \cup \boldsymbol{k}$. This replaces the keys for j and j' in $\boldsymbol{k}'$ with the keys in $\boldsymbol{k}$. $\mathcal{B}$ assigns one of its $\mathsf{Check}^{\mathrm{IR}}$ oracles to j and the other to j'.
3. $\mathcal{B}$ calls $\mathcal{A}$ with 1^d, 1^n, $\boldsymbol{k}''$, and $\{\boldsymbol{K}_i\}_{i \in I'}$ and simulates $\mathcal{A}$'s ownership queries as follows:
   - if $i \neq j$ and $i \neq j'$, then $\mathcal{B}$ has all the keys for $\boldsymbol{K}_i$, so $\mathcal{B}$ can emulate exactly the execution of $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_i)$, including keeping state.
   - If $i = j$ or $i = j'$, then $\mathcal{B}$ forwards the ownership query on to $\mathcal{B}$'s oracle for i.
4. When $\mathcal{A}$ returns K and K', $\mathcal{B}$ returns K and K'.

If $\mathcal{A}$ succeeds, then there is some pair $i, i' \in I - I'$ such that $\boldsymbol{K}_i \subseteq K$ and $\boldsymbol{K}_{i'} \subseteq K'$ and $K \cap K' = \emptyset$. Since j and j' were chosen uniformly at random, the probability that the unordered pair (i, i') is the same as the pair (j, j') is $\binom{n}{2}^{-1}$, so $\mathcal{B}$ succeeds with probability greater than $\frac{\binom{n}{2}/\binom{2d}{d}}{\binom{n}{2}} = \binom{2d}{d}^{-1}$. This contradicts Lemma 2.

So, contrary to the initial assumption, $\mathcal{A}$ must only be able to succeed with probability less than or equal to $\binom{n}{2}/\binom{2d}{d}$. □

Theorem 3 provides a way to determine the value of d for a given choice of probability ϵ_0 of a compromised requester violating Non-Separability with parameter ϵ_0 for n verifiers. Theorem 3 implies that $\epsilon_0 < \binom{n}{2}/\binom{2d}{d}$. Since $\binom{2d}{d} \geq 2^d$ for $d \geq 0$, this can be simplified to $\epsilon_0 < \binom{n}{2}/2^d$. Thus, it suffices to set d to $O(\log(\frac{n^2}{\epsilon_0}))$.

3.3. *Transferability of Atomic Signatures*

Idealized Random Keys provides a framework for proving that Atomic Signatures satisfies Transferability. We prove this in the form of a reduction, showing that Non-Separability of Idealized Random Keys with a given parameter ϵ implies Transferability of Atomic Signatures with the same parameter ϵ. Since Theorem 3 shows that Non-Separability holds for Idealized Random Keys (hence has a negligible parameter ϵ), it then follows that Transferability holds for Atomic Signatures.

Lemma 4. *If Idealized Random Keys satisfies Non-Separability, then Atomic Signatures satisfies Transferability.*

Proof. We prove the contrapositive by constructing an adversary $\mathcal{B}$ that violates Non-Separability of Idealized Random Keys using an adversary $\mathcal{A}$ that violates Transferability of Atomic Signatures. $\mathcal{B}$ is given keys $\boldsymbol{k}$ for Idealized Random Keys.

For each key $k \in \boldsymbol{k}$, $\mathcal{B}$ generates random values $z = \langle z_1, z_2, \ldots, z_{dn} \rangle$ to construct a set of keys to pass to $\mathcal{A}$ for Atomic Signatures. $\mathcal{B}$ will respond to verification oracle queries for $\mathcal{A}$; $\mathcal{B}$ does not answer signing queries, since $\mathcal{A}$ is not given a signing oracle (nor does it need one, since it has all the keys).

For verification queries on a message m, a tag τ, and an index j, $\mathcal{B}$ knows the signing keys, so $\mathcal{B}$ can check to see if $\mathsf{roweq}(m, \tau, (k, z))$ holds for each $(k, z) \in \boldsymbol{k}$. If $\mathsf{roweq}(m, \tau, (k, z))$ holds for all $(k, z) \in \boldsymbol{k}$, then $\mathcal{B}$ can return ∞; and if no instance of $\mathsf{roweq}(m, \tau, (k, z))$ holds for any $(k, z) \in \boldsymbol{k}$, then $\mathcal{B}$ can return 0.

But if some instances hold and others do not, then $\mathcal{B}$ does not know what to return, since $\mathcal{B}$ does not know which instances use keys in $\boldsymbol{K}_j$. One way to solve this problem would be for $\mathcal{B}$ to make ownership queries to $\mathsf{Check}^{\mathrm{IR}}$ for keys for some instances—knowing which verifiers own which keys for more keys gives $\mathcal{B}$ a higher probability of answering correctly. But if $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ returns $\perp$ on any such ownership query when $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ would not have returned $\perp$, then s_j gets set to $\perp$, and v_j is not set to $\perp$ in Atomic Signatures. In this case, $\mathcal{B}$ might not be able to violate Non-Separability using the message and tag that $\mathcal{A}$ returns to violate Transferability; for instance, $\mathcal{A}$ might return a message and tag that violate Transferability for j and some other verifier. And such a j could not be used to violate Non-Separability, since $s_j = \perp$ would hold. So,

$\mathcal{B}$ needs to ensure that if an ownership query to $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ causes $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ to return $\perp$ in the course of simulating a call to $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$, then $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ would also have returned $\perp$.

To gain more information about the values returned by $\mathsf{Ver}^{\mathrm{AS}}$, $\mathcal{B}$ proceeds as follows: $\mathcal{B}$ randomly generates n additional *known key pairs* $(k_1^*, z_1^*), (k_2^*, z_2^*), \ldots, (k_n^*, z_n^*)$ and assigns (k_j^*, z_j^*) to verifier j for $1 \leq j \leq n$. $\mathcal{B}$ adds these keys to $\boldsymbol{k}$, creating $\boldsymbol{k}'$, and creates n state variables $s_1, s_2, \ldots, s_n$, initializing each to 1. So, $\mathcal{B}$ will use $\mathcal{A}$ on security parameter $d+1$ to violate Non-Separability on security parameter d.[10]

1. $\mathcal{B}$ calls $A(1^{d+1}, 1^n, \boldsymbol{k}', \{\boldsymbol{K}_i\}_{i \in I'})$ and answers $\mathcal{A}$'s queries for m, τ, and j as follows:
   (a) *Initialization.* Set S^0 and S^1 to $\emptyset$.
   (b) *Check* s_j. If $s_j = \perp$, then return $\perp$.
   (c) *Key Discovery.* For each key pair (k, z) in $\boldsymbol{k}$, if $\mathsf{roweq}(m, \tau, (k, z))$ holds, then add k to set S^1; otherwise add k to set S^0.
   (d) *Check Opposite Keys.* If $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ holds, then iterate over each key k in S^0. Otherwise iterate over keys in S^1. Use oracle access to call $\mathsf{Check}^{\mathrm{IR}}(k, \boldsymbol{K}_j)$ on each such key k, and return $\perp$ if $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ ever returns $\perp$. Set s_j to $\perp$ when returning $\perp$.
   (e) If $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ holds, then return ∞. Otherwise, return 0.
2. When $\mathcal{A}$ returns m and τ, run Key Discovery as before to get S^0 and S^1. Return $K = S^1$ and $K' = S^0$.

It remains to show that this algorithm correctly simulates the operation of $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$ and that K and K' jointly violate Non-Separability. There are only three possible return values from the verification oracle $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j)$: 0, ∞, and $\perp$. We consider each case in turn:

- $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j) = 0$. In this case, the definition of $\mathsf{Ver}^{\mathrm{AS}}$ states that, for all key pairs (k, z) in $\boldsymbol{K}_j$, $\mathsf{roweq}(m, \tau, (k, z))$ will not hold. So, all of j's keys will be placed in S^0. Further, $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ will also fail to hold, so Check Opposite Keys will iterate over S^1. Thus, none of j's keys will be passed to $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$, so $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ will not return $\perp$, which means the simulation will not return $\perp$. Since $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ does not hold, the simulation returns 0, as required.
- $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j) = \infty$. In this case, the definition of $\mathsf{Ver}^{\mathrm{AS}}$ states that, for all key pairs (k, z) in $\boldsymbol{K}_j$, $\mathsf{roweq}(m, \tau, (k, z))$ holds. So, all of j's keys will be placed in S^1. Further, $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ must hold, so Check Opposite Keys will iterate over S^0. Thus, none of j's keys will be passed to $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$, so $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ will not return $\perp$, which means that the simulation will not return $\perp$. Since $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ holds, the simulation returns ∞, as required.
- $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j) = \perp$. The definition of $\mathsf{Ver}^{\mathrm{AS}}$ provides two cases in which a verifier might return $\perp$.

  First, it might be that $v_j \neq \perp$. In this case, the definition of $\mathsf{Ver}^{\mathrm{AS}}$ states that there must be some key pairs (k_r, z_r) and $(k_{r'}, z_{r'})$ in $\boldsymbol{K}_j$ such that $\mathsf{roweq}(m, \tau, (k_r, z_r))$

[10] The security parameter is now $d+1$ instead of d, because there are now $dn + n = (d+1)n$ keys.

holds and $\mathsf{roweq}(m, \tau, (k_{r'}, z_{r'}))$ does not hold. So, (k_r, z_r) is put in S^1, and $(k_{r'}, z_{r'})$ is put in S^0.

If $\mathsf{roweq}(m, \tau, (k_j^*, z_j^*))$ holds, then Check Opposite Keys will iterate over S^0, which contains $k_{r'}$, so $\mathsf{Check}^{\mathrm{IR}}(k_{r'}, \boldsymbol{K}_j)$ will be called and will return $\perp$. If not, then Check Opposite Keys will iterate over S^1, which contains k_r, so $\mathsf{Check}^{\mathrm{IR}}(k_r, \boldsymbol{K}_j)$ will be called and will return $\perp$. Either way, the simulation returns $\perp$, as required. And in both cases, the simulation sets s_j to $\perp$.

Second, it might be that $v_j = \perp$. By definition of $\mathsf{Ver}^{\mathrm{AS}}$, this means that $\mathsf{Ver}^{\mathrm{AS}}(\cdot, \cdot, \boldsymbol{K}_j)$ must have previously returned $\perp$. This means that s_j has already been set to $\perp$, by induction. So, the simulation returns $\perp$ in the step *Check* s_j, as required.

Thus, $\mathcal{B}$ simulates $\mathcal{A}$'s oracle calls correctly. When $\mathcal{A}$ succeeds, the definition of Transferability implies that there is some pair $j, j' \in I - I'$ such that the values of m and τ returned by $\mathcal{A}$ satisfy $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j) = \infty$ and $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_{j'}) = 0$. Therefore, $\mathsf{roweq}(m, \tau, (k, z))$ must hold for every $(k, z) \in K_j$ and for no $(k, z) \in K_{j'}$, which means that $\boldsymbol{K}_j \subseteq S^1 = K$ and $\boldsymbol{K}_{j'} \subseteq S^0 = K'$, as required. And $K \cap K' = \emptyset$ holds by definition. Values s_j and $s_{j'}$ are not set to $\perp$, since $\mathsf{Ver}^{\mathrm{AS}}(\cdot, \cdot, \boldsymbol{K}_j)$ and $\mathsf{Ver}^{\mathrm{AS}}(\cdot, \cdot, \boldsymbol{K}_{j'})$ never returned $\perp$. $\mathcal{B}$ succeeds with the same nonnegligible probability as $\mathcal{A}$. □

Note that the constructed adversary $\mathcal{B}$ in the proof of Lemma 4 needs the known key pairs to simulate the operation of the verification function. But known key pairs are not needed in the construction itself. The lack of known keys in the construction leads to the difference in security parameters between $\mathcal{B}$ and $\mathcal{A}$: the constructed adversary $\mathcal{B}$ needs d keys, but $\mathcal{A}$ needs $d + 1$.

The following theorem uses the previous results to show that Atomic Signatures is a Strong ∞-MVS scheme.

Theorem 5. *If the* MAC *is a pseudorandom function, then Atomic Signatures is a Strong ∞-MVS scheme.*

Proof. Lemma 1 shows that Atomic Signatures satisfies ∞-Completeness and Strong Unforgeability if the MAC is a pseudorandom function. And Lemma 4 and Theorem 3 together imply that Atomic Signatures satisfies Transferability. So, Atomic Signatures is a Strong ∞-MVS scheme. □

The reduction in Lemma 4 adds one key per verifier to the set of keys used in Atomic Signatures. So, the value $d = O(\log(\frac{n^2}{\epsilon_0}))$ computed in Sect. 3.2 using the probability ϵ_0 of a compromised signer being able to create a split tag gives a value of d that is 1 lower than the value needed for Atomic Signatures. But this does not affect the asymptotic complexity of the scheme: if we assume that computing the MAC of a message m takes time $O(|m|)$, then Atomic Signatures can be computed by generating a vector of dn MACs in time $O(|m|dn)$ and solving the factored matrix equation in time $O(n^2 d^2) = O(n^2 \log^2(\frac{n^2}{\epsilon_0}))$. So, the total asymptotic complexity of generating a tag is $O(|m|n \log(\frac{n^2}{\epsilon_0}) + n^2 \log^2(\frac{n^2}{\epsilon_0}))$.

4. Chain Signatures

Chain Signatures is a λ-MVS scheme that creates tags consisting of vectors of *subtags*. Each subtag contains the output of a MAC on the concatenation of previous subtags.

Key generation algorithm $\mathsf{Gen}^{\mathrm{CS}}(1^d, 1^n)$ produces n *known keys* $\boldsymbol{k}_1 = k_1^*, k_2^*, \ldots, k_n^*$ and dn *unknown keys* $\boldsymbol{k}_2 = \langle k_1, k_2, \ldots, k_{dn} \rangle$, where each key is an element of $\{0,1\}^b$. $\mathsf{Gen}^{\mathrm{CS}}(1^d, 1^n)$ then sets $\boldsymbol{k} = (\boldsymbol{k}_1, \boldsymbol{k}_2)$ and creates n vectors $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_n$. A vector $\boldsymbol{K}_j$ contains k_j^* as well as a set of d keys chosen uniformly at random from $\boldsymbol{k}_2$ such that vectors $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_n$ are disjoint.

$\mathsf{Sign}^{\mathrm{CS}}(m, \lambda, \boldsymbol{k})$ produces[11] a vector $\boldsymbol{C}^{\lambda,d}(m)$ consisting of λ *sections*, each divided into two *components*: we call component 1 the *known-key* component and component 2 the *unknown-key* component. Component 1 contains n subtags, and component 2 contains dn subtags, so each section contains $(d+1)n$ subtags. We write $\boldsymbol{C}^{\lambda,d}(m)[r,c,s]$ for the sth subtag in the cth component of the rth section of the tag generated by Chain Signatures for m and λ.[12] We use the natural lexicographic ordering on triples (r,c,s) used to index the subtags of Chain Signatures.[13] The value of a subtag is computed recursively as the MAC of the concatenation of m with the subtags in all previous components:

$$\boldsymbol{C}^{\lambda,d}(m)[r,c,s] \triangleq \mathsf{MAC}\Big(m \underset{(t,t',t'')<(r,c,1)}{||} \boldsymbol{C}^{\lambda,d}(m)[t,t',t''], \boldsymbol{k}_c[s]\Big). \tag{3}$$

Subtag s in component c of section r is said to be *supported* if the value of this subtag is identical to the MAC of the message and all previous components under key $\boldsymbol{k}_c[s]$. Figure 1 shows the structure of a signed message using Chain Signatures.

Verification algorithm $\mathsf{Ver}^{\mathrm{CS}}(\cdot,\cdot,\boldsymbol{K}_j)$ keeps state v_j to record whether it has ever been called with a message and tag that indicate that the signer is compromised. When called with a message m and tag τ, the verification algorithm checks all the subtags of

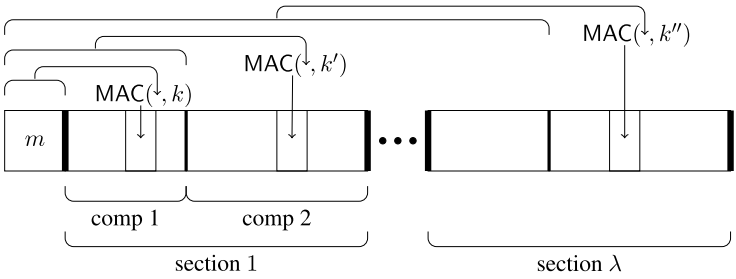

Fig. 1. The structure of a signed message using Chain Signatures.

[11] The definition of MVS schemes does not allow signing algorithm $\mathsf{Sign}^{\mathrm{CS}}(\cdot, \boldsymbol{k})$ to take three parameters. To get around this difficulty, when Chain Signatures is considered as a λ-MVS scheme for some $\lambda \in \mathbb{N}$, we define $\mathsf{Sign}^{\mathrm{CS}}(m, \boldsymbol{k})$ to mean $\mathsf{Sign}^{\mathrm{CS}}(m, \lambda, \boldsymbol{k})$.

[12] Note that we can compute the offset of $\boldsymbol{C}^{\lambda,d}(m)[r,c,s]$ in this tag as $(d+1)n(r-1)+(c-1)n+s-1$.

[13] Note that sections, components, and subtags indexed by our triples are numbered starting at 1 rather than the more customary value of 0.

j in τ to see if there is a supported subtag that follows a nonsupported subtag. If so, then verification concludes that the signer is compromised. And if not, then verification returns the value of the highest section in which it found a supported subtag.

More precisely, the verification algorithm works as follows, where λ' is the value of the highest section in which verification found a supported subtag for j, and c is the highest component in λ' in which a supported subtag was found:

- If $v_j = \perp$, then return $\perp$.
- Otherwise, if each known-key component below c contains exactly one supported subtag for j, and each unknown-key component below c contains d supported subtags for j, then return λ'.
- If no component contains supported subtags for j, then return 0.
- Otherwise, some component below c contains a nonsupported subtag for j, so return $\perp$ and set v_j to $\perp$.

Remark 4.1. The alternation of known-key and unknown-key components in tags generated by Chain Signatures is critical to the security of the algorithm. If a supported subtag t in one component follows a nonsupported subtag t' in another component, then there must be some pair of adjacent components c and c' such that c' comes before c, there is a nonsupported subtag in c', and there is a supported subtag in c. Since known-key and unknown-key components alternate, exactly one of c and c' must be a known-key component. So, whenever verification returns $\perp$, there is a known-key component and an adjacent unknown-key component that justify this return value.

A simpler—but wrong—version of Chain Signatures would not include known-key components. This would give a compromised signer an easy way to violate Transferability. For example, suppose that compromised signer i creates a tag τ in which all subtags in the first $\lambda - 1$ sections are supported. However, in section λ of tag τ, only one subtag is supported. The key for this subtag is owned by some verifier, say j. So, verification at j will return λ, and all other verifiers will return $\lambda - 1$. This reveals that j owns the key for this subtag. And i can perform this attack on each key to learn its attribution; i can create split tags once it knows the attribution of enough keys.

This attack fails in Chain Signatures due to the known-key components. Suppose that compromised signer i creates a tag τ in which all subtags in the first $\lambda - 1$ sections are supported. The first component in section λ is a known-key component. So, if i makes one of the subtags in the known-key component supported and all other subtags in section λ nonsupported, then j will return λ, and all other verifiers will return $\lambda - 1$. This reveals the attribution of j's known key. But the attribution of j's known key is known, so the adversary does not learn anything new about the keys.

Another variant on the same attack would be for the compromised signer to make all the subtags in the known-key component supported. But then, no matter which subtags are supported in the unknown-key component of section λ, all verifiers will return λ, since all find a supported subtag in this section.

A more complex variant of this attack would be to make some subtags in the known-key component supported and some nonsupported. Also, at least one subtag in the unknown-key component of section λ must be made supported (otherwise, the adversary learns nothing, as discussed above). But now there is some probability that there is

a verifier j' that has a nonsupported tag in the known-key component and a supported tag in the unknown-key component. This verifier will return $\perp$ on τ. So, if the adversary requests verification of τ from a verifier j', and j' does not return $\perp$, then the adversary learns that j' does not own the key used for the subtag in the unknown-keys.

Attacks in which some verifiers might return $\perp$ yield information to the adversary, but they also risk revealing to verifiers that the signer is compromised. If too many verifiers learn that the signer is compromised, then Transferability cannot be violated, since violations m and τ depend on verifiers j and j' such that $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j) \neq \perp$ and $\mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}) \neq \perp$. Our reduction below from Idealized Random Keys shows how to choose a value of d such that, with high probability, compromised signers never learn enough information to create split tags without revealing themselves as compromised to too many verifiers.

Remark 4.2. Chain Signatures as described above is expensive to compute; generating a tag costs $O(dn\lambda(|m| + dn\lambda))$, since inputs to the MAC grow linearly in the length of the tag. Figure 9 in Appendix D gives an algorithm that reduces the cost to $O(|m| + dn\lambda \log \lambda)$ using collision-resistant hash functions. Modified proofs of λ-Completeness, Unforgeability, and Non-Accusability follow the description of this more efficient version.

A λ-MVS scheme must satisfy λ-Completeness, Unforgeability, Non-Accusability, and Transferability. We prove that Chain Signatures is a λ-MVS scheme for $\lambda \in \mathbb{N}_{>0}$ using two lemmas. The first shows that Chain Signatures satisfies λ-Completeness, Unforgeability, and Non-Accusability; the second reduces Non-Separability of Ideal Random Keys to Transferability of Chain Signatures. Then, Theorem 3, along with the second lemma, implies that Chain Signatures satisfies Transferability.

Note that Chain Signatures does not satisfy Strong Unforgeability. Any adversary that receives a tag τ for message m that causes verifier j to return $\lambda > 1$ can produce a new tag τ' that causes verifier j to return $\lambda - 1$. All the adversary needs to do is to remove the last section, since tags for earlier sections do not depend on the last section. The previous $\lambda - 1$ sections consist entirely of supported subtags, so verifier j will return $\lambda - 1$ for m and τ'. But the adversary never received τ' from its signing oracle, so m and τ' together violate Strong Unforgeability.

Lemma 6. *For any $\lambda \in \mathbb{N}_{>0}$, if the* MAC *satisfies CTA Unforgeability, then Chain Signatures satisfies λ-Completeness, Unforgeability, and Non-Accusability.*

Proof. **λ-Completeness.** This follows directly from the definition: all subtags are supported by construction, so $\mathsf{Ver}^{\mathrm{CS}}(m, \mathsf{Sign}^{\mathrm{CS}}(m, \lambda, \boldsymbol{k}), \boldsymbol{K}_j) = \lambda$.

Unforgeability. We prove the contrapositive. Suppose that adversary $\mathcal{A}$ violates Unforgeability for some $I' \subseteq I$ with probability ϵ_0. We construct an adversary $\mathcal{B}$ that violates CTA Unforgeability of the MAC (for some key k') with probability ϵ_0/n. $\mathcal{B}$ chooses a key k_t^* uniformly at random from the n known keys and generates a new instance of Chain Signatures by calling $\mathsf{Gen}^{\mathrm{CS}}$ and replacing calls to $\mathsf{MAC}(\cdot, k_t^*)$ with (i) calls to $\mathcal{B}$'s MAC oracle when signing and (ii) $\mathcal{B}$'s verification oracle when verifying. When $\mathcal{A}$ succeeds, returning m and τ, the definition of Unforgeability states that there

is some $j \in I - I'$ for which $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j) > 0$. This means j must have (at least) a supported subtag in component 1 of section 1.

$\mathcal{B}$ returns m as its message and $\tau[1, 1, t]$ as its tag. With probability $1/n$, we have $t = j$, since t was chosen uniformly at random and independently of j. And $\mathsf{MAC}(m, k') = \tau[1, 1, t]$, because $\tau[1, 1, t] = \tau[1, 1, j]$ is the only subtag for j in component 1 of section 1, so it must be supported. The unique length of inputs to MACs for each component implies the only component for which $\mathcal{B}$ could have requested m from its MAC oracle is the very first. This request could only have been made if $\mathcal{A}$ requested m from its signing oracle, which does not occur by definition.

So, $\mathcal{B}$ never requested m from its MAC oracle, and $\mathcal{B}$ succeeds in violating CTA Unforgeability with probability ϵ_0/n.

Non-Accusability. We prove the contrapositive. Suppose that some adversary $\mathcal{A}$ violates Non-Accusability for some $I' \subseteq I$ with probability ϵ_0. Similar to the proof of Unforgeability, we construct a $\mathcal{B}$ that violates CTA Unforgeability of the MAC by building a new instance of Chain Signatures and calling $\mathcal{A}$. Instead of choosing a key at random from the known keys, however, $\mathcal{B}$ chooses a key k_t from the union of the known keys and the unknown keys. When $\mathcal{A}$ succeeds and returns m and τ, the definition of Non-Accusability states that there must be some j in $I - I'$ such that $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$, which means that there is some supported subtag for j that takes as input a nonsupported subtag for j in some component r.

With probability $1/((d + 1)n)$, key k_t was used to compute this supported subtag, since t was chosen uniformly at random and independently of the choice of the nonsupported subtag. In this case, $\mathcal{B}$ returns this supported subtag in component r as tag τ' and the message m concatenated with all components before component r as message m'. The length of m' means that it could only have been input to $\mathcal{B}$'s MAC oracle in component r. But since it contains a nonsupported subtag, it never would have been input to a MAC in $\mathcal{B}$'s simulation of the signing oracle, since only concatenations of supported subtags are input to the MAC oracle in $\mathcal{B}$'s simulation, by construction. So, m' has never been requested from $\mathcal{B}$'s MAC oracle.

Thus, m' and τ' violate CTA Unforgeability of the MAC with probability $\epsilon_0/((d + 1)n)$, which is nonnegligible. □

To prove that Chain Signatures satisfies Transferability, we reduce from Non-Separability of Idealized Random Keys. This reduction relies on the following characterization of tags: for each verifier j, message m, and tag pair τ, there is a highest known-key component of τ containing a supported subtag; we call this known-key component $\mathsf{highSup}(m, \tau, j, \boldsymbol{k})$. And there is a lowest known-key component containing a nonsupported subtag for j; we call this known-key component $\mathsf{lowNonSup}(m, \tau, j, \boldsymbol{k})$. Verification returns $\perp$ when a supported subtag follows a nonsupported subtag for a given verifier. As argued in Remark 4.1, at least one such pair of supported and nonsupported subtags in this case always involves a known-key component. The following lemma shows that these special known-key components indicate when to return $\perp$.

Lemma 7. *For verifier j, if the state v_j is not $\perp$, then $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$ if and only if there is, in τ, a nonsupported subtag for j in a component below $\mathsf{highSup}(m, \tau, j, \boldsymbol{k})$ or there is a supported subtag for j in a component above $\mathsf{lowNonSup}(m, \tau, j, \boldsymbol{k})$.*

Proof. The "if" direction trivially follows from the definition of $\mathsf{Ver}^{\mathrm{CS}}$: if a supported subtag for verifier j, message m, and tag τ follows a nonsupported subtag for j, then $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$.

We prove the "only if" direction by the contrapositive. Let λ^{HS} be set to $\mathsf{highSup}(m, \tau, j, \boldsymbol{k})$, and let λ^{LN} be set to $\mathsf{lowNonSup}(m, \tau, j, \boldsymbol{k})$. Suppose that all nonsupported subtags for j are in λ^{HS} or higher components—in fact, all nonsupported tags for j must be in higher components, since there is only one subtag for j in λ^{HS}, and this subtag is supported. Suppose further that all supported tags for j are in λ^{LN} or lower components—by the same argument as for λ^{HS}, all supported subtags for j must actually be in lower components than λ^{LN}.

We will show that $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ cannot return $\perp$. For the sake of contradiction, suppose that it does. Then, since v_j is not $\perp$, the definition of $\mathsf{Ver}^{\mathrm{CS}}$ states that there is a pair of keys k_1 and k_2 associated with j and components r^N and r^S such that $r^N < r^S$ holds, the subtag generated with k_1 in component r^N is not supported, and the subtag generated with k_2 in component r^S is supported. This happens because the verification algorithm returns $\perp$ only if a nonsupported subtag occurs in a lower component than a supported subtag.

By the argument above, $r^N > \lambda^{\mathrm{HS}}$ and $r^S < \lambda^{\mathrm{LN}}$ both hold. So, $\lambda^{\mathrm{HS}} < r^N < r^S < \lambda^{\mathrm{LN}}$ holds; see Fig. 2 for a depiction of these components. This means that there are at least two distinct components r^N and r^S between λ^{HS} and λ^{LN}, so one of the components between λ^{HS} and λ^{LN}, say r^K, must be a known-key component. Since $r^K < \lambda^{\mathrm{LN}}$ holds, the definition of λ^{LN} requires that j's subtag in r^K be supported. But, since $r^K > \lambda^{\mathrm{HS}}$ holds, the definition of λ^{HS} requires that j's subtag in r^K not be supported. This is a contradiction, since r^K only has one subtag for j, so $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ cannot return $\perp$. □

We now proceed to show the following lemma.

Lemma 8. *If Idealized Random Keys satisfies Non-Separability, then Chain Signatures satisfies Transferability.*

Proof. We prove the contrapositive: we construct an adversary $\mathcal{B}$ that violates Non-Separability of Idealized Random Keys using an adversary $\mathcal{A}$ that violates Transferability of Chain Signatures. The reduction for Transferability of Atomic Signatures relies on known keys that are added for the proof. For Chain Signatures, however, known keys are already part of the construction, so they do not need to be added for the proof.

$\mathcal{B}$ is given $\boldsymbol{k}$ for Idealized Random Keys and generates n keys to serve as the known keys for Chain Signatures. $\mathcal{B}$ then forms $\boldsymbol{k}'$ consisting of $\boldsymbol{k}$ and these known keys to

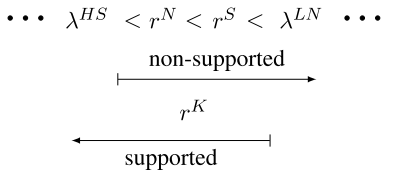

Fig. 2. Components used in the proof of Lemma 7.

pass to $\mathcal{A}$. In the following simulation, $\mathcal{B}$ keep state s_j for each simulated verifier j (as in the reduction for Atomic Signatures) and returns $\perp$ when $s_j = \perp$. Similarly, $\mathcal{B}$ sets s_j to $\perp$ when it returns $\perp$ for a query for verifier j.

Since $\mathcal{B}$ knows all the keys, $\mathcal{B}$ can check each subtag in each component to see if it is supported. $\mathcal{B}$ divides the keys into two sets for each component c based on whether or not the MAC using a key in c is supported; we call these sets *supported* and *non-supported*, respectively. $\mathcal{B}$ then uses the known keys to decide on which sets to call $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ to simulate a given call to $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$, as follows.

1. $\mathcal{B}$ finds $\mathsf{lowNonSup}(m, \tau, j, \boldsymbol{k})$ and calls $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ on all keys in all supported sets for higher components.
2. Similarly, $\mathcal{B}$ finds $\lambda = \mathsf{highSup}(m, \tau, j, \boldsymbol{k})$ and calls $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ on all keys in all nonsupported sets for lower components.
3. If any of the calls to $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ return $\perp$, then $\mathcal{B}$ returns $\perp$.
4. Otherwise, $\mathcal{B}$ returns λ, since verification for j returns the value of the highest section that contains a supported subtag for j.

This strategy simulates $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ perfectly. To see why, we consider the possible return values of $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$. When $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns 0, there are no supported subtags for j, so the simulation will also return 0.

When $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\lambda > 0$, there must be some supported subtag for j in section λ, and no supported subtags in higher sections. And since verification did not return $\perp$, all subtags for j in lower components must also be supported. Since the known-key subtag for j is the first subtag for j in section λ, it must also be supported; this means that the simulation will return λ.[14]

By Lemma 7, the simulation is also correct when $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$. Note that Lemma 7 implies that $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ in $\mathcal{B}$'s simulation will only return $\perp$ when $\mathsf{Ver}^{\mathrm{CS}}(\cdot, \cdot, \boldsymbol{K}_j)$ does, since the simulation calls $\mathsf{Check}^{\mathrm{IR}}(\cdot, \boldsymbol{K}_j)$ only on keys for subtags that match the description in the hypothesis of Lemma 7.

When $\mathcal{A}$ succeeds, returning a message m and tag τ, the definition of Transferability implies that there is some pair $j, j' \in I - I'$ and a section λ' such that $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j) = \lambda'$ and $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_{j'}) < \lambda' - 1$. $\mathcal{B}$ finds j, j', and λ' by simulating the verification function as before for each verifier. Then $\mathcal{B}$ returns the keys for the supported subtags in the unknown-key component of section $\lambda' - 1$ as K and the keys for the nonsupported subtags in the unknown-key component of section $\lambda' - 1$ as K'. This strategy always succeeds, since a violation of Transferability in section λ' for verifiers j and j' means that the subtags for j must be supported in the unknown-key component of section $\lambda' - 1$ and the subtags for j' must not be supported. Thus, $\mathcal{B}$ succeeds with the same probability as $\mathcal{A}$. □

Just as in the proof of Lemma 4 for Atomic Signatures, the adversary $\mathcal{B}$ in the proof of Lemma 8 depends critically on the known keys in its simulation of the verification oracle. But, unlike Atomic Signatures, the known keys are essential to the construction of Chain Signatures, as shown in Lemma 7.

[14] Note that no subtags in unknown components in section $\lambda + 1$ or higher could be supported, since the known subtag for j in section $\lambda + 1$ is not supported; having supported subtags in unknown components in section $\lambda + 1$ or higher would mean that verification would have to return $\perp$, not λ.

Remark 4.3. It might seem like a more natural construction for Chain Signatures would make the input to the MAC for each subtag be the concatenation of the message and all previous *subtags*, instead of the message and all subtags in previous *components*. But Lemma 7 no longer holds in this version of Chain Signatures. The problem is that a nonsupported subtag for some verifier j could be followed by a supported subtag for j in the same unknown-key component. In this case, there is no contradiction in the proof of Lemma 7. This means that the highest known-key component with a supported subtag and the lowest known-key component with a nonsupported subtag are not sufficient to simulate verification in the simpler version of Chain Signatures; Lemma 7 is critical in the reduction from Non-Separability of Idealized Random Keys to Transferability of Chain Signatures.

We are not aware of any attacks on this version of Chain Signatures despite the fact that the proof does not work. However, this version is less efficient than the version we prove secure, so we do not discuss it further.

The following theorem uses the previous results to show that Chain Signatures is a λ-MVS scheme.

Theorem 9. *For any $\lambda \in \mathbb{N}_{>0}$, if the* MAC *satisfies CTA Unforgeability, then Chain Signatures is a λ-MVS scheme.*

Proof. Lemma 6 shows that Chain Signatures satisfies λ-Completeness, Unforgeability, and Non-Accusability if the MAC satisfies CTA Unforgeability. Lemma 8 shows that Non-Separability of Idealized Random Keys implies Transferability of Chain Signatures. Since Theorem 3 shows that Non-Separability of Idealized Random Keys holds, it follows that Chain Signatures satisfies Transferability. □

The reduction in Lemma 8 shows that the value of the security parameter d is set to $d+1$ for Chain Signatures to have the same security as Idealized Random Keys has for d. However, the asymptotic complexity of d is the same, so the value $d = O(\log(\frac{n^2}{\epsilon_0}))$ computed in Sect. 3.2 using the probability ϵ_0 of a compromised signer being able to create a split tag is the same for Chain Signatures as for Idealized Random Keys. Using the running-time formula from Appendix D, we calculate that Chain Signatures can be generated in time $O(|m| + dn\lambda \log \lambda) = O(|m| + n\lambda \log(\frac{n^2}{\epsilon_0}) \log \lambda)$.

5. Performance

We implemented Atomic Signatures (AS) and Chain Signatures (CS) in C using OpenSSL 0.9.8e [23]. Using a hash function h, we compute a MAC for a message m and key k by setting $\mathsf{MAC}(m,k) = h(h(m) \mid\mid k)$, as suggested by Canetti et al. [6] for cases where many MACs must be computed for the same message. In our implementation, h is SHA-1 [28].[15] All shared keys comprise 160 bits, and the output of the MAC is

[15] Under the assumption that SHA-1 is pseudorandom, this MAC satisfies the properties required for our proof, according to Bellare et al. [3].

also 160 bits, so parameter $b = 160$.[16] We use all optimizations described in the paper and the appendices: pseudorandom functions are used to generate a factored matrix for Atomic Signatures, and hashing is used as in the pseudo-code of Fig. 9 in Appendix D to reduce the running time of Chain Signatures. The probability ϵ_0 that a compromised signer will be able to create a split tag is set to 2^{-64}, except where otherwise stated. Parameter λ is considered up to $\lambda = 3$, since this is a common value for protocols used in implementing distributed services.[17]

All tests were run on a 2.13-GHz Pentium M over Gentoo Linux kernel 2.6.22-gentoo-r9. RSA and DSA measurements were made for OpenSSL by running the commands `openssl speed rsa` and `openssl speed dsa` on this system. Each value represents a mean over 1000 runs; the error gives the sample standard deviation around this mean.

The performance of signature algorithms depends on three factors: the execution time for generating and checking tags, the tag size, and the key infrastructure required. Figures 3 and 4 show the execution time for generating and checking Chain Signatures. In Fig. 3, for $\lambda = 3$, Chain Signatures can generate tags faster than 1024-bit RSA for $n \leq 50$ and faster than 2048-bit RSA for all $n < 100$, which is more than sufficient for many applications. Figure 4 shows that checking Chain Signatures (for $\lambda = 3$ and

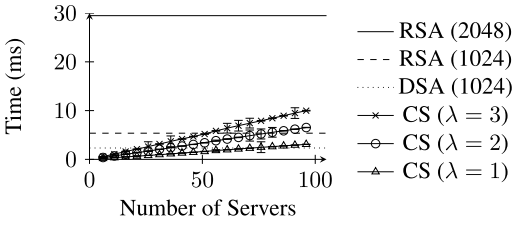

Fig. 3. Execution time for generating Chain Signatures ($\epsilon = 2^{-64}$).

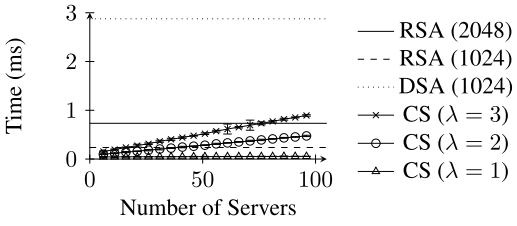

Fig. 4. Execution time for checking Chain Signatures ($\epsilon = 2^{-64}$).

[16] Atomic Signatures requires that an adversary only be able to violate Strong Unforgeability with a given probability ϵ_0'. The proof of Lemma 1 bounds ϵ_0' by $\text{poly}(d, n)/2^b$, but the exact value of the polynomial factor $\text{poly}(d, n)$ depends on Lemmas 10–12, which provide asymptotic, rather than concrete, bounds. So, we instead choose b to satisfy $\epsilon_0' < 1/2^b$, since the polynomial factor will make only a small difference in the choice of b for small values of n and d.

[17] Note that the probability of a compromised signer generating a split tag is statistical rather than computational; the proofs of Non-Separability of Idealized Random Keys rely only on the randomness of key distribution and not on any computational assumption.

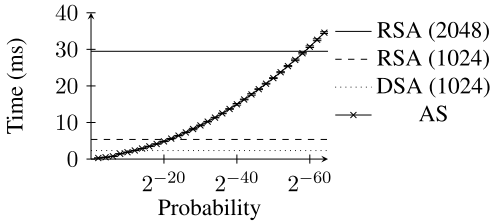

Fig. 5. Execution time to generate Atomic Signatures for six verifiers and different probabilities of generating a split tag.

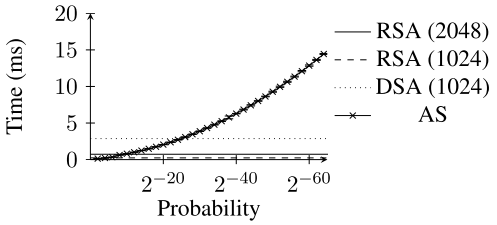

Fig. 6. Execution time to check Atomic Signatures for six verifiers and different probabilities of generating a split tag.

$\epsilon = 2^{-64}$) is faster than 2048-bit RSA for $n < 75$. Higher probabilities of split tags may be acceptable in some contexts and lead to faster generation and checking of signatures.

Atomic Signatures costs $O(d^2n^2)$, so tags that use many random keys are more expensive to generate; the efficiency depends on the probability that a signer can generate split tags. Figures 5 and 6 show how generating and checking times vary for six verifiers and different probabilities of creating a split tag. Atomic Signatures can generate tags for six verifiers faster than 2048-bit RSA for probabilities down to about 2^{-55}. But checking tags generated by Atomic Signatures is more expensive for probabilities below about 2^{-25}.

Even though execution time for generating and checking tags based on the schemes in this paper is sometimes lower than RSA and DSA, tag size for our signature algorithms is significantly larger. Chain Signatures and Atomic Signatures require significant space even for small n, since the size of the signature depends linearly on d. For instance, in Chain Signatures with six verifiers, $\epsilon = 2^{-64}$, and $\lambda = 3$, generating signatures takes about 581 µs, which is fast, but the size of a tag is 13680 bytes. These sizes are acceptable in circumstances where signature transfer time is negligible—for instance, between processes in operating systems, or across local-area networks using Gigabit Ethernet switches.

Key-management infrastructure costs for Chain Signatures and Atomic Signatures are also relatively high, since each verifier must store $O(dn)$ keys. For instance, with $n = 4$ and $\epsilon = 2^{-64}$, each verifier must share $d = 36$ keys with the signer. And if $n = 36$ with the same value of ϵ, then d becomes 40. Rekeying requires that signer i not learn with which verifier it shares a given key. If the keys for a single verifier j were replaced without replacing keys for other verifiers, then i would learn which of its keys

correspond to j. Even if keys for some subset of the verifiers were replaced, then signer i would gain some information about which keys correspond to which verifiers. Thus, all keys must be replaced simultaneously.

These performance results show that in some contexts, MVS schemes have comparable, and sometimes even better, performance than public-key signature schemes. Unlike these schemes, however, MVS schemes are proven secure only assuming the existence of pseudorandom functions, whereas these public-key signature schemes are only known to be secure in the heuristic random oracle model. The results of our experiments show that it is possible to have provable security and efficiency for signature schemes.

6. Related Work

Many authentication schemes use symmetric message authentication codes and try to achieve properties similar to public-key signature schemes. But none is able to handle an unbounded number of adaptive queries. We succeed by using a unusual secret-key setup along with state kept by verifiers. Previous work achieves different properties.

λ-Limited Transferability Chaum and Roijakkers [9] were the first to suggest constructing tags that could be transferred a finite number of times. Their scheme allows signed messages to be transferred only once. Pfitzmann and Waidner [24] followed with a construction, called *pseudosignatures*, that is somewhat similar to Chain Signatures: it creates tags that can be transferred an arbitrary fixed number of times. Both the work of Chaum and Roijakkers and Pfitzmann and Waidner provide unconditional security.

Like MVS schemes, pseudosignatures depend on a secret-key setup; multiple keys are shared between the signer and each verifier, and the signer cannot attribute keys to verifiers. However, pseudosignature tag size is directly proportional to the number of queries an adversary can submit to a verification oracle. Even if pseudosignatures were implemented with computationally-secure MACs, they would only be able to tolerate a fixed number of verification queries.

Arbitrary Transferability with Unconditional Security Many schemes have been proposed for tags that are both unconditionally secure and can be transferred an arbitrary number of times. For instance, recent work [18,27,29] generalizes Multi-Receiver Authentication (MRA) codes (invented by Desmedt et al. [11]) to unconditionally-secure polynomial codes that satisfy similar properties to Transferability. These constructions are called *MRA*^{3} *codes*. MRA^3 codes constrain the number of signing and verification oracle queries as well as the number of possible signatures that a signer can create, since each signature leaks information.

Johansson [19] proposes a different authentication scheme that also satisfies unconditional security properties; it is similar in form to Atomic Signatures: signers and verifiers each have secret keys that are used to solve a matrix equation. But unlike Atomic Signatures, each signature in Johansson's scheme provides a set of linear equations over the signer's secret keys, so keys must be refreshed after a fixed number of signatures.

Computational Security Other schemes similar to MVS have been designed for particular protocols in the computational model. For instance, MACs are sometimes considered shared-key signatures, despite not satisfying Transferability. And in some fault-tolerant distributed systems (e.g., Practical Byzantine Fault Tolerance (PBFT) [7]), vectors of MACs are used to improve protocol speed over public-key signatures.

Aiyer et al. [1] present schemes in which servers use MACs to generate tags having similar properties to public-key signatures. Unlike public-key signatures and MVS schemes, however, their construction relies on communication between clients and servers to produce and verify signed messages. And they also require that no more than $1/3$ of the servers in the system be compromised.

In a distributed setting where at most t signers may be compromised, Lamport [20] suggests (in a set of slides on Byzantine Paxos) collecting $(\lambda + 1)t + 1$ tags from different signers. A signer in the scheme creates λ vectors consisting of n subtags each, where each subtag of each vector contains a MAC of all the vectors before it, along with the message. This scheme does not provide adaptive security, since an adversary with oracle access to the signing functionality can create a split tag by the following procedure. The adversary requests a tag for m and receives τ. Then the adversary corrupts τ to τ' by overwriting some subtags with random strings and requests a tag for $m \,||\, \tau'$, receiving a tag τ''. The tag $\tau' \,||\, \tau''$ is split for m, since all subtags in τ'' are supported, but some subtags in τ' are not supported.

Canetti et al. [6] propose a multicast MAC scheme that is closely related to the schemes in this paper. In this scheme, a collection of keys is associated with each verifier; keys are chosen randomly from a large set. Signers create a tag for a message m by generating a MAC of m for each key they know. The algorithm distributes keys at random with probability $\frac{1}{t+1}$ if up to t verifiers may collude to try to forge tags. Keys in this protocol may thus be shared by more than one verifier.[18] Canetti et al. show that given $\epsilon > 0$, having $e(t+1)\log(\frac{1}{\epsilon})$ keys in total suffices to guarantee that tags can be forged only with probability less than ϵ. However, these tags do not satisfy Transferability, since an adversary can create a new tag from a correctly signed tag by corrupting one subtag. This new tag will be accepted by some verifiers and not by others.

7. Summary

This paper shows that MVS schemes provide lower-cost authentication than public-key signature schemes while guaranteeing similar properties. Achieving these properties in Atomic Signatures and Chain Signatures requires implementing a specialized secret-key setup. The setup encodes an asymmetric relationship between the signer and verifiers, since verifiers know which keys are owned by which signers, but signers do not know which keys are owned by which verifiers; the proof of Transferability depends critically on this asymmetry. Asymmetry in knowledge about keys thus appears to be fundamental for achieving Transferability, both in public-key signature schemes and MVS schemes.

[18] Atomic Signatures and Chain Signatures are closely related in key distribution to Canetti et al., but each key is only shared between a pair of servers, so compromised relays are forced to guess keys to forge tags for messages.

Acknowledgements

We thank Michael Clarkson, Andrew Myers, Robbert van Renesse, Kevin Walsh, the Cornell Security Discussion Group, and the Microsoft Research Crypto Lunch Seminar for discussions about this research. We thank Andrew Myers, Robbert van Renesse, and Draga Zec for comments on an earlier version of this paper.

Appendix A. Lemmas for Strong Unforgeability of Atomic Signatures

We prove several lemmas that together simplify the proof of Strong Unforgeability of Atomic Signatures to the case where all but one verifier are compromised, no verifier queries are allowed for an adversary, and $\mathsf{MAC}(\cdot, k)$ is replaced by a random function $v_k(\cdot)$.

Lemma 10. *If Atomic Signatures satisfies Strong Unforgeability when all but one verifier are compromised, then Atomic Signatures satisfies Strong Unforgeability.*

Proof. Suppose $\mathcal{A}$ violates Strong Unforgeability using an arbitrary set $I' \subseteq I$ of compromised verifiers. We construct an adversary $\mathcal{B}$ that violates Strong Unforgeability when all but one verifier, say verifier j, are compromised. $\mathcal{B}$ is given the keys for all verifiers but j and is given oracle access to a verification oracle for j as well as a signing oracle. $\mathcal{B}$ maps its verifiers randomly to verifiers in the simulation for $\mathcal{A}$; this does not change the view of $\mathcal{A}$, since keys are chosen uniformly at random. $\mathcal{B}$ runs $\mathcal{A}$ and simulates its oracle queries as follows.

1. $\mathcal{B}$ calls $\mathcal{A}(1^d, 1^n, \{\boldsymbol{K}_i\}_{i \in I'})$.
2. When $\mathcal{A}$ makes a signing oracle query, $\mathcal{B}$ passes the query to its signing oracle and returns its response.
3. When $\mathcal{A}$ makes a verification oracle query for verifier j', $\mathcal{B}$ calls its verification oracle if $j' = j$ and otherwise uses its knowledge of the keys for j' to perform verification for j' and return the result.
4. When $\mathcal{A}$ returns m and τ, $\mathcal{B}$ checks that $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_j) \neq 0$. If so, then $\mathcal{B}$ returns m and τ, and otherwise, $\mathcal{B}$ aborts.

When $\mathcal{A}$ returns m, τ, there is some $j' \in I - I'$ such that $\mathsf{Ver}^{\mathrm{AS}}(m, \tau, \boldsymbol{K}_{j'}) \neq 0$, and there is a $1/n$ chance that $j = j'$, since the position of j in $I - I'$ was chosen independently of the view of $\mathcal{A}$. So, $\mathcal{B}$ succeeds with probability ϵ/n if $\mathcal{A}$ succeeds with probability ϵ. □

Note that Lemma 10 is not entirely trivial: the act of compromising more verifiers does not necessarily make it easier for the adversary to violate Strong Unforgeability, since the adversary must find a noncompromised verifier that will accept its forged tag. So, the availability of more noncompromised verifiers might in principle make the adversary's task easier.

Lemma 11. *If Atomic Signatures satisfies Strong Unforgeability when no verifier queries are allowed to an adversary and all but one verifier are compromised, then*

Atomic Signatures satisfies Strong Unforgeability when all but one verifier are compromised.

Proof. Given an adversary $\mathcal{A}$ that succeeds with nonnegligible probability with polynomial bound $p(n)$ on its number of verification queries, we can construct a new adversary $\mathcal{B}$ that succeeds with nonnegligible probability without using any verifier queries. Assume, without loss of generality, that $\mathcal{A}$ always makes a verifier query for the pair m and τ that it outputs.[19]

$\mathcal{B}$ simulates verifier queries from $\mathcal{A}$ for a message–tag pair m, τ as follows: if m has been requested already from the signing oracle, which returned τ, then return ∞. If m has not been requested from the signing oracle, or the signing oracle returned anything but τ, then return 0. $\mathcal{B}$ stores each verification query. When $\mathcal{A}$ outputs m and τ, $\mathcal{B}$ chooses one of the stored verification queries m', τ' uniformly at random and outputs it.

$\mathcal{A}$ either uses more than one verification query (the final one), or it does not. If it does not, then $\mathcal{B}$ succeeds every time $\mathcal{A}$ does, since $\mathcal{B}$ always chooses the one query that $\mathcal{A}$ made. And this is the value that $\mathcal{A}$ returned.

If $\mathcal{A}$ uses more than one verification query, then either some verification queries (other than the last) that were not received from the signing oracle should have returned a value other than 0, or all verification queries (other than the last) that were not received from the signing oracle should have returned 0. If all except the last should return 0, then $\mathcal{B}$ succeeds only when it returns the m and τ from the last query. $\mathcal{A}$ succeeds with nonnegligible probability ϵ and makes at most $p(n)$ queries, so $\mathcal{B}$ succeeds, in this case, with probability greater than or equal to $\epsilon / p(n)$, which is nonnegligible.

If some queries not received from the signing oracle should return a value other than 0, then there is no guarantee about the success probability of $\mathcal{A}$, since $\mathcal{B}$ no longer simulates all of the verification queries correctly. But there is still a maximum bound of $p(n)$ on the number of verification queries, and there is at least one query that caused $\mathcal{B}$ to fail to simulate the verification queries correctly. The definition of $\mathcal{B}$ guarantees that this query violates Strong Unforgeability, since the query was not received from the signing oracle. This means that the probability of $\mathcal{B}$ returning a query m and τ that violates Strong Unforgeability is at least $1/p(n)$, which is nonnegligible.

So, $\mathcal{B}$ always succeeds in violating Strong Unforgeability with nonnegligible probability. □

For simplicity in stating the next lemmas, call the version of Atomic Signatures in the hypothesis of Lemma 11 *verifier-free* Atomic Signatures, and call Atomic Signatures when no verifier queries are allowed, all but one verifier are compromised, and $\mathsf{MAC}(\cdot, k)$ is replaced by a random function $v_k(\cdot)$ *verifier-free random* Atomic Signatures.

Lemma 12. *If* MAC *is a pseudorandom function, then if verifier-free random Atomic Signatures satisfies Strong Unforgeability, then verifier-free Atomic Signatures satisfies Strong Unforgeability.*

[19] If this is not the case, then there is another adversary $\mathcal{D}$ that succeeds with the same probability as $\mathcal{A}$ but performs the extra query. $\mathcal{D}$ runs $\mathcal{A}$, and when $\mathcal{A}$ returns m and τ, $\mathcal{D}$ queries m and τ from the verification oracle before returning them.

Proof. Suppose, on the contrary, that verifier-free random Atomic Signatures satisfies Strong Unforgeability, but verifier-free Atomic Signatures does not. This means that there is an adversary $\mathcal{A}$ that succeeds with nonnegligible probability ϵ when interacting with oracles that use $\mathsf{MAC}(\cdot, k)$ to answer signing queries, and, by assumption, succeeds with only negligible probability ϵ' when interacting with oracles that use random functions $v_k(\cdot)$ to answer signing queries.

Now construct a sequence of hybrids as follows: H_i uses $v_{k_j}(\cdot)$ instead of $\mathsf{MAC}(\cdot, k_j)$ for keys k_j such that $1 \leq j \leq i$, then uses $\mathsf{MAC}(\cdot, k_j)$ for the remaining keys k_j such that $i+1 \leq j \leq dn$. Note that H_0 is verifier-free Atomic Signatures, and H_{dn} is verifier-free random Atomic Signatures.

A standard hybrid argument shows that there must be an i such that $\mathcal{A}$ succeeds with nonnegligible probability on hybrid H_i and succeeds with negligible probability on hybrid H_{i+1}. But then there exists an algorithm $\mathcal{D}$ that can distinguish a random function from MAC, since the only difference between hybrids H_i and H_{i+1} is that H_i uses MAC in its $i+1$st position, whereas H_{i+1} uses a random function in this position.

$\mathcal{D}$ sets up an instance of the hybrid Atomic Signatures scheme using its oracle (which is either a pseudorandom or a random function) in the $(i+1)$st position. Then $\mathcal{D}$ calls $\mathcal{A}$ on this instance and returns 1 if $\mathcal{A}$ succeeds and 0 if $\mathcal{A}$ fails. Since $\mathcal{A}$ succeeds with nonnegligible probability when there is a pseudorandom function in the $(i+1)$st position and succeeds only with negligible probability when there is a random function in the $(i+1)$st position, $\mathcal{D}$ succeeds in distinguishing pseudorandom from random functions with nonnegligible probability. This contradicts the hypothesis that MAC is a pseudorandom function. □

Appendix B. Noncompromised Signers

The constructions of MVS schemes in this paper allow the signer to be compromised. But there are places where it is reasonable to make stronger assumptions. For instance, when it is sound to assume that the signer is not compromised, we can simplify our constructions significantly. This assumption holds in some common contexts: for example, in operating systems, the OS itself is trusted by the processes and sometimes signs messages (e.g., capabilities) to processes.

When the signer is not compromised, Transferability can be weakened to the following:

Weak Transferability. For every nonuniform PPT adversary $\mathcal{A}$, there exists a negligible function ϵ such that for any choice of $I' \subseteq I$,

$$\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big);$$
$$(m, \tau) \leftarrow A^{\mathsf{Sign}(\cdot,\boldsymbol{k}),\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, \{\boldsymbol{K}_i\}_{i\in I'}\big):$$
$$\big(\exists j, j' \in I - I' \;:\; \big|\mathsf{Ver}(m, \tau, \boldsymbol{K}_j) - \mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'})\big| > 1\big)\Big] \leq \epsilon(d, n).$$

Weak Transferability implies that even an adversary that controls an arbitrary subset of verifiers and has signing and verification oracles (for the other verifiers) cannot produce message and tag pair on which two correct verifiers will produce values that differ by more than one. Notice that the adversary in this case does not control the signer.

We call a λ-MVS scheme that satisfies λ-Completeness, Unforgeability, and Weak Transferability a *Weak λ-MVS* scheme.

B.1. *Known-Key Atomic Signatures*

Known-Key Atomic Signatures (KA) is a Weak ∞-MVS scheme based on Atomic Signatures, and it only uses one key for each verifier. KA follows exactly the algorithms for Atomic Signatures for the case $d = 1$. But this means that verifiers can never return $\perp$, since either their single instance of (2) is satisfied, or it is not.

Theorem 13. *If the* MAC *is a pseudorandom function, then Known-Key Atomic Signatures is a Weak ∞-MVS scheme.*

Proof. **∞-Completeness.** Same reasons as Atomic Signatures.

Strong Unforgeability. This follows from exactly the same proof as for Atomic Signatures. The only difference is that the Union Bound does not include a factor of d, since each verifier only has 1 key rather than d.

Weak Transferability. We show that Strong Unforgeability implies Weak Transferability. As we have already shown that Known-Key Atomic Signatures satisfies Strong Unforgeability, this implies that it also satisfies Weak Transferability.

We prove the contrapositive. Suppose that there is an adversary $\mathcal{A}$ that can violate Weak Transferability with nonnegligible probability ϵ. $\mathcal{A}$ uses access to a signing oracle and verification oracles to produce a message m and tag τ such that for some pair $j, j' \in I - I'$, it holds that $|\mathsf{Ver}^{\mathrm{KA}}(m, \tau, \boldsymbol{K}_j) - \mathsf{Ver}^{\mathrm{KA}}(m, \tau, \boldsymbol{K}_{j'})| > 1$. This means that one verifier must return ∞ and the other must return 0. Without loss of generality, assume that j returns ∞ and j' returns 0.

We now produce an adversary $\mathcal{B}$ that violates Strong Unforgeability. $\mathcal{B}$ is given the same signing and verification oracles as $\mathcal{A}$ and must produce a message and tag that it has never received from the signing oracle but causes some verifier $j \in I - I'$ to produce a value that is not 0. $\mathcal{B}$ simply calls $\mathcal{A}$, simulates $\mathcal{A}$'s oracle calls by passing them to $\mathcal{B}$'s oracles, and returns the values of m and τ returned by $\mathcal{A}$.

Since verification for j returns ∞, the values m and τ will suffice to violate Strong Unforgeability as long as m and τ were never received from $\mathcal{B}$'s signing oracle. But the same values of m and τ also cause $j' \in I - I'$ to return 0. And ∞-Completeness implies that no message and tag returned from the signing oracle ever cause a correct verifier to return 0. So, m and τ were not received from the signing oracle, and $\mathcal{B}$ succeeds in violating Strong Unforgeability. So, Strong Unforgeability implies Weak Transferability.

And since we proved above that Known-Key Atomic Signatures satisfies Strong Unforgeability, it follows that Known-Key Atomic Signatures also satisfies Weak Transferability. □

B.2. *Known-Key Chain Signatures*

Known-Key Chain Signatures (KC) is a Weak λ-MVS scheme obtained by simplifying Chain Signatures—it does not use unknown-key components in tags. So, each verifier shares exactly one key with the signer. Its algorithms operate as follows

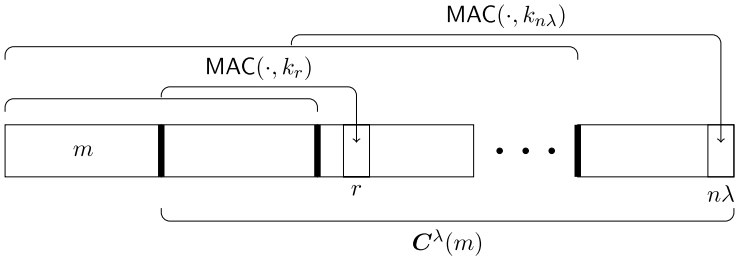

Fig. 7. The structure of Known-Key Chain Signatures.

- $\mathsf{Gen}^{\mathrm{KC}}(1^n)$ simply sets up pairwise shared keys. The signer is given a vector $\boldsymbol{k}$ of keys, and each verifier j is given $\boldsymbol{k}[j]$.
- $\mathsf{Sign}^{\mathrm{KC}}(m, \lambda, \boldsymbol{k})$ performs exactly the same operations as in Chain Signatures but only uses the known-key components. We index subtag s in section r by a pair (r, s) with the natural lexicographic ordering. This subtag is computed for the tag $\boldsymbol{C}^{\lambda}(m)$ as follows:

$$\boldsymbol{C}^{\lambda}(m)[r,s] \triangleq \mathsf{MAC}\Big(m \underset{(t,t')<(r,1)}{||} \boldsymbol{C}^{\lambda,d}(m)[t,t'], \boldsymbol{k}[s]\Big). \tag{4}$$

- $\mathsf{Ver}^{\mathrm{KC}}(m, \tau, \boldsymbol{K}_j)$ finds the highest section λ for which j's subtag is supported, and returns λ. If there is no such section, then it returns 0. Verification never returns $\perp$, since the signer cannot be compromised.

Figure 7 shows the structure of KC, where we write k_p for $\boldsymbol{k}[p \bmod n]$.

Generating a signature requires $n\lambda$ steps: the tag contains $n\lambda$ subtags, and each step produces one subtag by computing the MAC of a vector that is of size at most $|m| + n\lambda$. Thus the total cost of generating a tag is $O(n\lambda(|m| + n\lambda))$. The total cost can be significantly reduced, as explained in Appendix D.

Theorem 14. *For any $\lambda \in \mathbb{N}_{>0}$, if the* MAC *satisfies CTA Unforgeability, then KC is a Weak λ-MVS scheme.*

Proof. **λ-Completeness.** This follows trivially from the definition of KC, just as for Chain Signatures.

Unforgeability. The proof is the same as for Chain Signatures when the adversary causes some verifier to return a value in $\mathbb{N}_{>0}$: an adversary $\mathcal{B}$ is constructed that violates CTA Unforgeability of the MAC.

Weak Transferability. The proof is almost identical to the proof of Non-Accusability of Chain Signatures: if an adversary violates Weak Transferability, then by definition, some subtag will be supported that takes as input a subtag that is not supported. The probability of success for our construction in this case, however, is $\frac{\epsilon}{n}$ rather than $\frac{\epsilon}{(d+1)n}$, since the constructed adversary that violates CTA Unforgeability of the MAC guesses a key k_t from a set of size n rather than $(d+1)n$. □

Appendix C. Impossibility of Avoiding Split Tags

Transferability asserts that, except with negligible probability, even compromised signers cannot find split tags. But for public-key signatures, it is impossible to have split tags, since all verifiers use the same function to check tags. A natural question is whether such perfect transferability can be achieved in our MVS setting.

We can prove that an ∞-MVS scheme must contain a public-key signature scheme to avoid split tags perfectly. Start with an ∞-MVS scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ that satisfies ∞-Completeness, Unforgeability, and Non-Accusability. Now strengthen Transferability to the following.

Perfect Transferability. For any PPT $\mathcal{A}$ and for any choice of $I' \subseteq I$,

$$\begin{aligned}
&\Pr\Big[\big(\boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I}\big) \leftarrow \mathsf{Gen}\big(1^d, 1^n\big);\\
&\quad (m,\tau) \leftarrow A^{\{\mathsf{Ver}(\cdot,\cdot,\boldsymbol{K}_i)\}_{i\in I-I'}}\big(1^d, 1^n, \boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I'}\big):\\
&\quad \big(\exists j, j' \in I - I' \;:\; \mathsf{Ver}(m,\tau,\boldsymbol{K}_j) \neq \perp \wedge \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'}) \neq \perp\\
&\qquad\qquad \wedge \big|\mathsf{Ver}(m,\tau,\boldsymbol{K}_j) - \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'})\big| > 1\big)\Big] = 0.
\end{aligned}$$

Note that the only difference between the definition of Transferability (see page 314) and Perfect Transferability is that $\epsilon(d,n)$ is set to 0 in the definition of Perfect Transferability.

Perfect Transferability implies that there are no split tags under any choice of keys, because if a split tag existed, then an adversary that guessed message and tag pairs at random would have some nonzero probability of choosing it. Also note that we can remove the restriction on verifiers j and j' being noncompromised: if there is any pair of verifiers for which a split tag can be created, then an adversary can choose not to compromise those verifiers and guess the message and split tag with nonzero probability. We can thus rewrite Perfect Transferability as follows:

$$\begin{aligned}
\forall d, n, \forall \boldsymbol{k}, \{\boldsymbol{K}_i\}_{i\in I} \in \mathrm{Range}\big(\mathsf{Gen}\big(1^d, 1^n\big)\big), \forall m, \tau, \forall j, j' \in I \;:\;&\\
\big(\mathsf{Ver}(m,\tau,\boldsymbol{K}_j) \neq \perp \wedge \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'}) \neq \perp\big) \implies&\\
\big|\mathsf{Ver}(m,\tau,\boldsymbol{K}_j) - \mathsf{Ver}(m,\tau,\boldsymbol{K}_{j'})\big| \leq 1.&
\end{aligned}$$

To simplify the statement of Perfect Transferability further, we must consider a restricted class of ∞-MVS schemes.

Given any ∞-MVS scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, we can define a new ∞-MVS scheme $\Sigma' = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}')$, in which the verifier only returns 0, ∞, and $\perp$. To do so, $\mathsf{Ver}'(m,\tau,\boldsymbol{K}_j)$ calls $\mathsf{Ver}(m,\tau,\boldsymbol{K}_j)$ and gets a reply v. Then Ver' returns v if v is one of 0, ∞, or $\perp$. Otherwise, Ver' returns 0. Note that Ver' satisfies ∞-Completeness if Ver does, since ∞-Completeness for Σ guarantees that Ver always returns ∞ on message and tag pairs generated by Sign. And it is trivial to see that Σ' satisfies each of Unforgeability, Non-Accusability, Strong Unforgeability, Perfect Transferability, and Transferability if Σ does. We call Σ' a *normalized* ∞-MVS scheme from Σ.

Since verifiers that do not return $\perp$ in a normalized ∞-MVS scheme either return 0 or ∞, and $|\infty - 0| > 1$ holds, Perfect Transferability can be rewritten for normalized ∞-MVS schemes as follows:

$$\forall d, n, \forall \boldsymbol{k}, \{\boldsymbol{K}_i\}_{i \in I} \in \text{Range}\big(\mathsf{Gen}\big(1^d, 1^n\big)\big), \forall m, \tau, \forall j, j' \in I \ : \\ \big(\mathsf{Ver}(m, \tau, \boldsymbol{K}_j) \neq \perp \wedge \mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}) \neq \perp\big) \implies \\ \mathsf{Ver}(m, \tau, \boldsymbol{K}_j) = \mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}).$$

Perfect Transferability interacts with Unforgeability, since verifiers in a normalized ∞-MVS scheme that satisfies Perfect Transferability can all simulate the actions of other verifiers perfectly; a tag that violates Unforgeability must do so for all verifiers at once.

For public-key signature schemes, the property corresponding to Unforgeability is Chosen Message Attack (CMA) security [17]. A public-key signature scheme is secure under CMA if no adversary $\mathcal{A}$ can produce a message m and tag τ that cause the verification algorithm to return ∞, even if $\mathcal{A}$ can see tags for messages of its choice. Naturally, as in Unforgeability, the adversary cannot return a message it requested from its signing oracle.

The following theorem says that any normalized ∞-MVS scheme that satisfies Perfect Transferability instead of Transferability is effectively a public-key signature scheme secure under CMA [17].[20] The intuition behind the theorem is that Perfect Transferability effectively makes all verifiers use the same algorithm: access to one verifier allows perfect simulation of the actions of any other verifier.

Given a normalized ∞-MVS scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ that satisfies ∞-Completeness, Unforgeability, Non-Accusability, and Perfect Transferability, we can define a public-key signature scheme $\Sigma_j^D = (\mathsf{Gen}^D, \mathsf{Sign}^D, \mathsf{Ver}^D)$ as follows for any j in I. Let $\mathsf{Gen}^D(1^d)$ call $\mathsf{Gen}(1^d, 1^n)$ to produce public key $K = \boldsymbol{K}_j$ and secret key $k = \boldsymbol{k}$. Then $\mathsf{Sign}^D(\cdot, k)$ just calls $\mathsf{Sign}(\cdot, \boldsymbol{k})$. And $\mathsf{Ver}^D(\cdot, \cdot, K)$ calls $\mathsf{Ver}(\cdot, \cdot, \boldsymbol{K}_j)$, getting response v. Ver^D returns v if v is 0 or ∞, and Ver^D returns 0 if v is $\perp$, since all verifiers use the same verification function in Σ_j^D, hence never disagree about its return value.

Theorem 15. *If* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is a normalized* ∞*-MVS scheme satisfying* ∞*-Completeness, Unforgeability, Non-Accusability, and Perfect Transferability, then, for any* $j \in I$, Σ_j^D *is a public-key signature secure under CMA.*

Proof. By the contrapositive. Suppose that we are given a nonuniform PPT adversary $\mathcal{A}$ that violates CMA security of Σ_j^D with nonnegligible probability ϵ.

We will construct a PPT $\mathcal{B}$ that violates Unforgeability of the MVS scheme for any $I' \subseteq I$ such that $j \in I'$ and $\exists j' \in I - I'$, as follows. $\mathcal{B}$ is given $1^d, 1^n, \{\boldsymbol{K}_i\}_{i \in I'}$, and oracle access to $\mathsf{Sign}(\cdot, \boldsymbol{k}) = \mathsf{Sign}^D(\cdot, k)$ and $\{\mathsf{Ver}(\cdot, \cdot, \boldsymbol{K}_i)\}_{i \in I - I'}$, so in particular, $\mathcal{B}$ knows $\boldsymbol{K}_j$. $\mathcal{B}$ proceeds as follows:

- $\mathcal{B}$ calls $\mathcal{A}(1^d, \boldsymbol{K}_j)$

[20] Note that there is a normalized ∞-MVS scheme for every ∞-MVS scheme.

  - When $\mathcal{A}$ requests $\mathsf{Sign}^D(m, k)$, $\mathcal{B}$ calls the signing oracle $\mathsf{Sign}(m, \boldsymbol{k})$ and returns its response.
  - $\mathcal{A}$ evaluates $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j)$ to compute $\mathsf{Ver}^D(m, \tau, K)$.
- When $\mathcal{A}$ returns m and τ, $\mathcal{B}$ returns m and τ.

When $\mathcal{A}$ succeeds, m and τ satisfy $\mathsf{Ver}(m, \tau, \boldsymbol{K}_j) = \mathsf{Ver}^D(m, \tau, K) = \infty$. In this case, Perfect Transferability and Non-Accusability, taken together, imply that $\mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}) = \infty$ with all but negligible probability: Non-Accusability implies that $\mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}) \neq \perp$ with all but negligible probability, and Perfect Transferability (in its rewritten form for normalized ∞-MVS schemes) states that in this case, $\mathsf{Ver}(m, \tau, \boldsymbol{K}_{j'}) = \mathsf{Ver}(m, \tau, \boldsymbol{K}_j) = \infty$ always holds. This violates Unforgeability. $\mathcal{B}$ has never requested m from its signing oracle, because $\mathcal{A}$ is required by assumption never to request m of its signing oracle. $\mathcal{B}$ succeeds with the same nonnegligible probability ϵ as $\mathcal{A}$.

Thus, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver}(\cdot, \cdot, \boldsymbol{K}_j))$ is a public-key signature secure under CMA if (Gen, Sign, Ver) is a normalized ∞-MVS scheme that satisfies ∞-Completeness, Unforgeability, Non-Accusability, and Perfect Transferability. □

Appendix D. Efficient Chain Signatures

To make Chain Signatures and Known-Key Chain Signatures more efficient, we employ a different implementation that uses a family of collision-resistant hash functions to keep the size of the input to the MAC constant. The algorithm for generating tags using Known-Key Chain Signatures is presented in Fig. 8. There, h is chosen from H, a family of collision-resistant hash functions, operator $||$ is concatenation as before, and we define $x \,||\, y = x$ if $y = \texttt{NULL}$. Key generation algorithm Gen provides the chosen h to the signer and each verifier. To generate a tag, a signer follows the same algorithm as before, except that the input to the MAC in a given section is now the section number, along with the hash of the concatenation of two values: (1) the input to the previous section and (2) the hash of the previous section.

To check a subtag in section p, a verifier must use each subtag in each section that precedes section p and build up $w_p(m)$, the input to the MACs in section p. Verifiers follow the algorithm in Fig. 8 to build up $w_p(m)$ and use it to compute the MACs corresponding to the subtags they are checking.

To calculate the time needed to compute a tag, we assume that both the hash and the MAC execute in time linear in the length of their input. We also assume that both the hash and the MAC produce a constant-size output.

The loop over p in Fig. 8 has λ iterations, and each iteration involves a hash of a value of constant length, followed by a loop with n iterations and a hash computation over data of size $O(n)$. Since p has size $\log \lambda$ and $w_p(m)$ has constant size, the loop over p' takes time $O(n \log \lambda)$. There is also an initial cost of time $O(|m|)$ to compute $w_1(m)$. So, the total time to generate Known-Key Chain Signatures is $O(|m| + \lambda + n\lambda \log \lambda + n) = O(|m| + n\lambda \log \lambda)$.

This more efficient algorithm for Known-Key Chain Signatures is generalized to Chain Signatures in Fig. 9. Similar to Known-Key Chain Signatures, the input to the MAC for a given component is a number indexing the component, along with the hash

$$
\begin{array}{l}
w_0(m) := m; \\
v_0(m) := \texttt{NULL}; \\
\text{for } p := 1 \text{ to } \lambda \\
\quad w_p(m) := h\big(w_{p-1}(m) \mid\mid v_{p-1}(m)\big); \\
\quad \text{for } p' := 1 \text{ to } n \\
\quad\quad \boldsymbol{C}^{\lambda}(m)[p, p'] := \mathsf{MAC}\big(p \mid\mid w_p(m), k_p\big); \\
\quad v_p(m) := h\left(\mathop{||}_{t=1}^{n} \boldsymbol{C}^{\lambda}(m)[p, t]\right)
\end{array}
$$

Fig. 8. The hashing version of Known-Key Chain Signatures.

$$
\begin{array}{l}
w'_0(m) := m; \\
v'_0(m) := \texttt{NULL} \\
\text{for } p := 1 \text{ to } \lambda \\
\quad w_p(m) := h\big(w'_{p-1}(m) \mid\mid v'_{p-1}(m)\big) \\
\quad \text{for } p' := 1 \text{ to } n \\
\quad\quad \boldsymbol{C}^{\lambda,d}(m)[p, 1, p'] := \mathsf{MAC}\big(2(p-1) \mid\mid w_p(m), \boldsymbol{k}_0[p']\big) \\
\quad v_p := h\left(\mathop{||}_{t=1}^{n} \boldsymbol{C}^{\lambda,d}(m)[p, 1, t]\right) \\
\quad w'_p(m) := h\big(w_p(m) \mid\mid v_p(m)\big) \\
\quad \text{for } p' := 1 \text{ to } dn \\
\quad\quad \boldsymbol{C}^{\lambda,d}(m)[p, 2, p'] := \mathsf{MAC}\big((2(p-1)+1) \mid\mid w'_p(m), \boldsymbol{k}_1[p']\big) \\
\quad v'_p(m) := h\left(\mathop{||}_{t=1}^{dn} \boldsymbol{C}^{\lambda,d}(m)[p, 2, t]\right)
\end{array}
$$

Fig. 9. The hashing version of Chain Signatures.

of the concatenation of two values: (1) the input to the previous component and (2) the hash of the previous component.

We can calculate the running time of the algorithm of Fig. 9 as follows. The loop over p has λ iterations. And each iteration has a hash over data of constant size, followed by a loop with n iterations (each performing a MAC of data of size $O(\log \lambda)$) and a hash of data of size $O(n)$. Then there is a hash of data of constant size, a loop with dn iterations (each performing a MAC of data of size $O(\log \lambda)$), and a hash of data of length $O(dn)$. And, as before, there is an initial cost of $O(|m|)$ to compute $w_1(m)$. So, the time needed to compute Chain Signatures using the algorithm of Fig. 9 is $O(|m| + \lambda(n \log \lambda + n + dn \log \lambda + dn)) = O(|m| + dn\lambda \log \lambda)$.

To use the algorithms of Figs. 8 and 9 in Known-Key Chain Signatures and Chain Signatures, we must modify the proofs of Unforgeability and Non-Accusability for Chain Signatures, and Weak Transferability for Known-Key Chain Signatures, since arguments based on unique input sizes to the MACs of each section no longer work. Instead, the unique prefix p in the computation of the value $\mathsf{MAC}(p \mid\mid w_p(m), k_p)$ in the algorithm guarantees that the signing oracle would only have performed a given computation for a subtag in the pth section.

In these new versions of Chain Signatures and Known-Key Chain Signatures, we say that a subtag is *supported* if it is identical to the MAC of the hash value using w_p or w'_p

defined recursively in Figs. 8 and 9 over all previous components and the message. So, a verifier can determine if its subtags are supported by computing the hashes of previous components and the message and computing the MAC of this value.

Lemma 16. *If* MAC *satisfies CTA Unforgeability and H is a family of collision-resistant hash functions, then Chain Signatures using the algorithm described in Fig. 9 satisfies λ-Completeness, Unforgeability, and Non-Accusability.*

Proof. **λ-Completeness.** As before, the proof of λ-Completeness follows by construction: signing and verification use the same algorithms to generate and check tags, so the evaluation of the verification function $\mathsf{Ver}^{\mathrm{CS}}(m, \mathsf{Sign}^{\mathrm{CS}}(m, \lambda, \boldsymbol{k}), \boldsymbol{K}_j)$ returns λ for any λ and any choice of j.

Unforgeability. We prove the contrapositive. Suppose that adversary $\mathcal{A}$ violates Unforgeability for some $I' \subseteq I$ with probability ϵ. We construct an adversary $\mathcal{B}$ that attempts to violates CTA Unforgeability of the MAC (for some key k') or collision-resistance of the family H. $\mathcal{B}$ is given MAC and VF oracles and is given the description of a hash function h chosen randomly from H.

$\mathcal{B}$ chooses a key k_t^* uniformly at random from the n known keys and generates a new instance of Chain Signatures by calling $\mathsf{Gen}^{\mathrm{CS}}$ and replacing calls to $\mathsf{MAC}(\cdot, k_t^*)$ with calls to $\mathcal{B}$'s MAC oracle when signing and $\mathcal{B}$'s verification oracle when verifying. $\mathcal{B}$ uses h as its hash function in the execution of signing and verification. When $\mathcal{A}$ succeeds, returning m and τ, the definition of Unforgeability states that there is some $j \in I - I'$ for which $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j) > 0$. This means j must have (at least) a supported subtag in component 1 of section 1.

$\mathcal{B}$ returns $0 \,||\, h(m)$ as its message and $\tau[1, 1, t]$ as its tag. With probability $1/n$, it holds that $t = j$, since t was chosen uniformly at random and independently of j. And $\mathsf{MAC}(0 \,||\, h(m), k') = \tau[1, 1, t]$ in this case, because $\tau[1, 1, t] = \tau[1, 1, j]$ is the only subtag for j in component 1 of section 1, so it must be supported. The prefix 0 in the MAC computation guarantees that this MAC could only have been computed for the first component of the first section. There are two possible cases.

In the first case, $\mathcal{A}$ requested some $m' \neq m$ from its signing oracle such that $h(m) = h(m')$. Then $0 \,||\, h(m) = 0 \,||\, h(m')$, so this message was requested of the MAC oracle, and CTA Unforgeability is not violated. But m' and m are a collision for the hash function, so $\mathcal{B}$ returns m and m' and violates collision resistance of H.

In the second case, $\mathcal{A}$ did not request any m' such that $h(m) = h(m')$, so $\mathcal{B}$ never requested $0 \,||\, h(m)$ from its MAC oracle, since $\mathcal{A}$ never requested m, by assumption. So, $\mathcal{B}$ succeeds in violating CTA Unforgeability.

So, either $\mathcal{B}$ violates CTA Unforgeability or returns a collision. And at least one case must occur with nonnegligible probability if $\mathcal{A}$ succeeds with nonnegligible probability. So, $\mathcal{B}$ succeeds with nonnegligible probability, and Chain Signatures satisfies Unforgeability.

Non-Accusability. We prove the contrapositive. Suppose that some adversary $\mathcal{A}$ violates Non-Accusability for some $I' \subseteq I$ with probability ϵ. Similar to the proof of Unforgeability, we construct a $\mathcal{B}$ that attempts to violates CTA Unforgeability of the MAC and collision resistance of the family H by building a new instance of Chain Signatures and calling $\mathcal{A}$. Instead of choosing a key at random from the known keys, however, $\mathcal{B}$

chooses a key k_t from the union of the known keys and the unknown keys. When $\mathcal{A}$ succeeds and returns m and τ, the definition of Non-Accusability states that there must be some j in $I - I'$ such that $\mathsf{Ver}^{\mathrm{CS}}(m, \tau, \boldsymbol{K}_j)$ returns $\perp$, which means that there is some supported subtag for j in a component r that takes as input a nonsupported subtag for j. Without loss of generality, let the component for the nonsupported subtag for j immediately precede the component for the supported subtag for j. And let this be the lowest position in the tag at which a supported subtag for j in one component follows a nonsupported subtag for j in the previous component.

There is a $1/((d+1)n)$ probability that k_t is the key used to compute this supported subtag, since t was chosen uniformly at random and independently of choice of the nonsupported subtag. Suppose that the nonsupported subtag for t is in component 2 of section $r-1$, followed by a supported subtag for t in component 1 of section r (the same argument applies for a nonsupported subtag in component 1 of some section r followed by a supported subtag in component 2 of section r, but the indices differ accordingly). $\mathcal{B}$ returns as a message m' the input for the supported subtag in component 1 of section r: $2(r-1) \mid\mid h(w'_{r-1}(m) \mid\mid v''_{r-1}(m))$, where $w'_{r-1}(m)$ is the normal computed value for component 2 of section $r-1$ in the signing algorithm of Fig. 9, and $v''_{r-1}(m) = h(||_{p=1}^{dn} \tau[r-1, 2, p])$. This is the normal algorithm for computing the v' value in the pseudo-code, but v'' contains a nonsupported subtag for j from τ. $\mathcal{B}$ returns as its tag τ' the corresponding supported subtag for j in component 1 of section r.

By the definition of Non-Accusability, when $\mathcal{A}$ succeeds, the MAC of the message m' returned by $\mathcal{B}$ is the value of the tag τ' returned by $\mathcal{B}$. So, the only question is whether or not m' was ever requested from the MAC oracle. If m' was never requested from the MAC oracle, then the proof is complete.

So, assume that m' was requested from the MAC oracle. Since m' starts with $2(r-1)$, it could only be requested from the MAC oracle in an execution of the signing algorithm for component 1 of section r. There are two possible cases.

In the first case, m' was requested in an execution of the signing algorithm for message m (the message on which m' and τ' are based). Note, however, that m' was formed by taking the hash of a nonsupported subtag, and this cannot occur in the execution of the signing algorithm for m; the nonsupported subtag must differ from the supported subtag that would be computed in section $r-1$ for m. So, the only way that m' could have been requested in the signing algorithm for m is if there were a collision in the hash function, either in the outer hash in computing m' or the hash in computing v''.

In the second case, m' was requested in an execution of the signing algorithm for some message m'' that is not equal to m. Note that m' contains the output of the hash on the concatenation of $w'_{r-1}(m)$ with the hash of an adversarially chosen value. For m' to be input to the MAC oracle in the computation of component 1 of section r during the signing of m'', it would have to be the case that $h(w'_{r-1}(m) \mid\mid v''_{r-1}(m)) = h(w'_{r-1}(m'') \mid\mid v'_{r-1}(m''))$ holds. And w' is constructed from concatenations of hashes of its input message (m or m'' in this case) with other values. Since m and m'' differ, and the two values $h(w'_{r-1}(m) \mid\mid v''_{r-1}(m))$ and $h(w'_{r-1}(m'') \mid\mid v'_{r-1}(m''))$ have identical lengths and recursive structure, the only way that these two hash values could be the same is if there were a hash collision. In both cases, $\mathcal{B}$ can find the hash collision by comparing all the hashes computed during the simulation.

Thus, either m' and τ' violate CTA Unforgeability of the MAC with nonnegligible probability, or a collision is found with nonnegligible probability. So, $\mathcal{B}$ succeeds with nonnegligible probability, and Chain Signatures satisfies Non-Accusability. □

As before, Weak Transferability of Known-Key Chain Signatures follows from a parallel argument to Non-Accusability of Chain Signatures, but with a success probability of ϵ/n instead of $\epsilon/(d+1)n$.

References

[1] A.S. Aiyer, L. Alvisi, R.A. Bazzi, A. Clement, Matrix signatures: From MACs to digital signatures in distributed systems, in *Proc. 22nd International Symposium on Distributed Computing*. Lecture Notes in Computer Science, vol. 5218 (Springer, Berlin, 2008), pp. 16–31

[2] M. Bellare, J. Kilian, P. Rogaway, The security of cipher block chaining, in *Advances in Cryptology—CRYPTO'94, Proc. 14th Annual International Cryptology Conference*, ed. by Y. Desmedt. Lecture Notes in Computer Science, vol. 839 (Springer, Berlin, 1994), pp. 341–358

[3] M. Bellare, O. Goldreich, A. Mityagin, The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, 2004. Report 2004/309, 2004

[4] D. Boneh, G. Durfee, M. Franklin, Lower bounds for multicast message authentication, in *Advances in Cryptology—Eurocrypt'01, Proc. International Conference on the Theory and Applications of Cryptographic Techniques*, ed. by B. Pfitzmann. Lecture Notes in Computer Science, vol. 2045 (Springer, Berlin, 2001), pp. 437–452

[5] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing. *J. Cryptol.* **17**(4), 297–319 (2004)

[6] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas. Multicast security: A taxonomy and some efficient constructions, in *IEEE INFOCOM'99. Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2 (IEEE Computer Society, Los Alamitos, 1999), pp. 706–716

[7] M. Castro, B. Liskov, Practical Byzantine fault tolerance, in *Proc. 3rd Symposium on Operating Systems Design and Implementation*, Berkeley, CA, February 1999. USENIX Association, Co-sponsored by IEEE TOCS and ACM SIGOPS, pp. 173–186

[8] L.S. Charlap, H.D. Rees, D.P. Robbins, The asymptotic probability that a random biased matrix is invertible. *Discrete Math.* **82**(2), 153–163 (1990)

[9] D. Chaum, S. Roijakkers, Unconditionally secure digital signatures, in *Advances in Cryptology—CRYPTO'90, Proc. 10th Annual Cryptology Conference*, ed. by A. Menezes, S.A. Vanstone. Lecture Notes in Computer Science, vol. 537 (Springer, Berlin, 1990), pp. 206–214

[10] R. Cramer, V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack, in *Advances in Cryptology—CRYPTO'98, Proc. 18th Annual International Cryptology Conference*, ed. by H. Krawczyk. Lecture Notes in Computer Science, vol. 1462 (Springer, Berlin, 1998), pp. 13–25

[11] Y. Desmedt, Y. Frankel, M. Yung. Multi-receiver/multi-sender network security: efficient authenticated multicast/feedback, in *IEEE INFOCOM'92: Proc. 11th Annual Joint Conference of the IEEE Computer and Communications Societies on One World through Communications* (IEEE Computer Society, Los Alamitos, 1992), pp. 2045–2054

[12] W. Diffie, M.E. Hellman, New directions in cryptography. *IEEE Trans. Inf. Theory* **IT-22**(6), 644–654 (1976)

[13] D. Dolev, The Byzantine Generals strike again. *J. Algorithms* **3**(1), 14–30 (1982)

[14] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, *Reasoning about Knowledge* (MIT Press, Cambridge, 2003)

[15] U. Feige, A. Shamir, Witness indistinguishable and witness hiding protocols, in *STOC'90: Proc. 22nd Annual ACM Symposium on Theory of Computing*, ed. by H. Ortiz (ACM, New York, 1990), pp. 416–426

[16] O. Goldreich, *Foundations of Cryptography*, vol. I (Cambridge University Press, Cambridge, 2001)

[17] S. Goldwasser, S. Micali, R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988)

[18] G. Hanaoka, J. Shikata, Y. Zheng, H. Imai, Unconditionally secure digital signature schemes admitting transferability, in *Advances in Cryptology—Asiacrypt'00, Proc. 6th International Conference on the Theory and Application of Cryptology and Information Security*, ed. by T. Okamoto. Lecture Notes in Computer Science, vol. 1976 (Springer, Berlin, 2000), pp. 130–142

[19] T. Johansson, Further results on asymmetric authentication schemes. *Inf. Comput.* **151**(1–2), 100–133 (1999)

[20] L. Lamport, Personal communication

[21] M. Marsh, F.B. Schneider, CODEX: A robust and secure secret distribution system. *IEEE Trans. Dependable Secure Comput.* **1**(1), 34–47 (2004)

[22] M. Naor, M. Yung, Universal one-way hash functions and their cryptographic applications, in *STOC'89: Proc. 21st Annual ACM Symposium on Theory of Computing* (ACM, New York, 1989), pp. 33–43

[23] OpenSSL Project, Available at http://www.openssl.org

[24] B. Pfitzmann, M. Waidner, Unconditional Byzantine agreement for any number of faulty processors, in *Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science*, ed. by A. Finkel, M. Jantzen. Lecture Notes in Computer Science, vol. 577 (Springer, Berlin, 1992), pp. 339–350

[25] R.L. Rivest, A. Shamir, L. Adelman, A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **21**(2), 120–126 (1978)

[26] J. Rompel, One-way functions are necessary and sufficient for secure signatures, in *STOC'90: Proc. 22nd Annual ACM Symposium on Theory of Computing* (ACM, New York, 1990), pp. 387–394.

[27] R. Safavi-Naini, L. McAven, M. Yung, General group authentication codes and their relation to "Unconditionally-secure signatures", in *Public Key Cryptography—PKC 2004, Proc. 7th International Workshop on Theory and Practice in Public Key Cryptography*, ed. by F. Bao, R.H. Deng, J. Zhou. Lecture Notes in Computer Science, vol. 2947 (Springer, Berlin, 2004), pp. 231–247

[28] FIPS 180-1. Secure Hash Standard. Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington, DC, April 1995

[29] J. Shikata, G. Hanaoka, Y. Zheng, H. Imai, Security notions for unconditionally secure signature schemes, in *Advances in Cryptology—Eurocrypt'02, Proc. 21st International Conference on the Theory and Applications of Cryptographic Techniques*, ed. by L.R. Knudsen. Lecture Notes in Computer Science, vol. 2332 (Springer, Berlin, 2002), pp. 434–449

[30] L. Zhou, F.B. Schneider, R. van Renesse, COCA: A secure distributed online certification authority. *ACM T. Comput. Syst.* **20**(4), 329–368 (2002)