Digital Signatures

6.1600 Course Staff

Fall 2023

In the last section, our strategy for authentication depended on two parties sharing a secret key. In that discussion, we completely left out of the picture how these parties should exchange this secret key. Our implication was that they went to some private room and exchanged the key in secret, but in many cases this is not practical: if they could whisper a key, why not just whisper the message?

Luckily, there is a way to get around this requirement for a shared secret using *public-key cryptography*.[1]

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

[1] Whitfield Diffie and Martin E Hellman. "New Directions in Cryptography". In: *Transactions on Information Theory* 22.6 (1976).

The original Diffie-Hellman paper from 1976, which introduced public-key cryptography, is a fascinating read.

1 *Definitions*

The basic idea of public-key cryptography, applied to authentication, is that each party will generate two linked keys—a secret signing key and a public verification key. The verification key will be good enough to verify that a signature is valid, but not to generate new signatures.

Definition 1.1 (Signature Scheme)**.** A signature scheme is associated with a message space $\mathcal{M}$ and three efficient algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

- $\mathsf{Gen}(\lambda) \to (\mathsf{sk}, \mathsf{vk})$. The key-generation algorithm as input a security parameter $\lambda \in \mathbb{N}$ and outputs a secret signing key sk and public verification key vk. The algorithm Gen runs in time $\mathrm{poly}(\lambda)$.
- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$. The signing algorithm takes as input a secret key sk and a message $m \in \mathcal{M}$, and outputs a signature σ.
- $\mathsf{Ver}(\mathsf{vk}, m, \sigma) \to \{0,1\}$. The signature-verification algorithm takes as input a public verification key vk, a message $m \in \mathcal{M}$, and a signature σ, and outputs $\{0,1\}$, indicating acceptance or rejection.

In theoretical papers, people will write $\mathsf{Gen}(1^\lambda)$ to indicate that the key-generation algorithm takes as input a length-λ string of ones. This is just a hack to make the input given to Gen λ bits long so that the Gen algorithm can run in time polynomial in its input length: $\mathrm{poly}(\lambda)$. If we express λ in binary, then $\mathsf{Gen}(\lambda)$ gets a $\log_2 \lambda$-bit input and can only run in time $\mathrm{poly}(\log \lambda)$. This distinction is really unimportant, but if you see the 1^λ notation, you will now know what it means.

For a signature scheme to be useful, a correct verifier must always accept messages from an honest signer. Formally, we have:

Definition 1.2 (Digital signatures: Correctness)**.** A digital-signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is *correct* if, for all messages $m \in \mathcal{M}$:

$$\Pr\left[\mathsf{Ver}(\mathsf{vk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1 \colon (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)\right] = 1.$$

The standard security notion for digital signatures is very similar to that for MACs (**??**). The only difference here is that a digital-signature scheme splits the single secret MAC key into two keys:

a secret signing key and a public verification key. Otherwise the definition is essentially identical.

Definition 1.3 (Digital signatures: Security – existential unforgeability under chosen message attack)**.** A digital-signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is *secure* if all efficient adversaries win the following security game with only negligible probability:

- The challenger runs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)$ and sends vk to the adversary.
- For $i = 1, 2, \ldots$ (polynomially many times)
  - The adversary sends a message $m_i \in \mathcal{M}$ to the challenger.
  - The challenger replies with $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, m_i)$.
- The adversary outputs a message-signature pair (m^*, σ^*).
- The adversary wins if $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m^* \notin \{m_1, m_2, \ldots\}$.

Notice that this security definition does not guarantee that an attacker cannot forge a new signature on a message that it has already seen a signature of. Namely, given a valid message-signature pair (m, σ) an adversary may be able to produce additional valid message-signature pairs on the same message: $(m, \sigma'), (m, \sigma''), \ldots$.

In some applications, we want to prohibit an attacker from finding *any* new message-signature pair. We call this security notion "*strong* existential unforgeability under chosen message attack." The definition is the same as in Definition 1.3 except that we require the adversary to find a valid-message signature pair (m^*, σ^*) such that $(m^*, \sigma^*) \notin \{(m_1, \sigma_1), (m_2, \sigma_2), \ldots\}$. Standard digital-signature schemes, such as the elliptic-curve digital signature algorithm (EC-DSA) or the RSA algorithm with full-domain hashing (RSA-FDH), are believed to have this strong security property.

2 *Constructing a Signature Scheme*

In the following sections, we will show how to construct a digital-signature scheme from any one-way function (**??**).

We will generate a signature scheme that is secure, but that has relatively large signatures and public keys: to achieve security against attackers running in time 2^λ, this signature scheme has signatures of $O(\lambda^2)$ bits. Widely used modern digital signature schemes (e.g., EC-DSA) have signatures of $O(\lambda)$ bits.

We will construct this scheme in three stages:

1. Construct a signature scheme for signing a *single bit*.
2. Construct a *one-time secure* signature scheme for signing a *fixed length* messages. With this scheme, an attacker who sees two

One benefit of the signature scheme that we present here is that—unlike EC-DSA, RSA, DSA, and other widely used signature schemes—this one is plausibly secure even against *quantum* adversaries. There is ongoing work to standardize signature schemes secure against quantum adversaries; see `https://csrc.nist.gov/projects/pqc-dig-sig`

signatures under the same signing key can forge signatures. In addition, the secret signing key for this scheme will be larger than the size of the message being signed.

3. Construct a *one-time secure* scheme for *arbitrary-length messages*. Here, we construct a one-time signature scheme whose secret signing key is independent of the length of the signed message.

4. Construct a *many-time secure* scheme (i.e., a fully secure one under Definition 1.3) for *arbitrary-length messages*. This last scheme is a fully secure and fully functional digital-signature scheme.

3 Constructing a Signature Scheme for Signing a Single Bit

This signature scheme is not useful on its own, and is given only as a step towards the final construction. It uses as a building block a one-way function $f : \mathcal{X} \to \mathcal{Y}$. Recall that f a one-way function if it is easy to compute but hard to invert; namely there is an efficient algorithm that given $x \in \mathcal{X}$ outputs $f(x)$, and at the same time any efficient algorithm given $y = f(x)$ for a random $x \leftarrow \mathcal{X}$, finds an inverse $x' \in \mathcal{X}$ such that $f(x') = f(x)$ with only negligible probability.

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{vk})$. Choose two random elements x_0, x_1 from $\mathcal{X}$ and let $(y_0, y_1) = (f(x_0), f(x_1))$. Output $\mathsf{sk} = (x_0, x_1)$ and $\mathsf{vk} = (y_0, y_1)$.
- $\mathsf{Sign}(\mathsf{sk}, b) \to \sigma$. Parse $\mathsf{sk} = (x_0, x_1)$ and output $\sigma = x_b$,
- $\mathsf{Ver}(\mathsf{vk}, b, \sigma) \to \{0, 1\}$. Parse $\mathsf{vk} = (y_0, y_1)$ and output 1 if and only if $f(\sigma) = y_b$. (Otherwise, the signing routine rejects.)

In this construction, we leave the security parameter λ implicit. To be fully formal, Gen would take λ an input. The one-way function f and its domain $\mathcal{X}$ would both depend on λ. So we would write f_λ and $\mathcal{X}_\lambda$.

4 One-time-secure Signatures (Lamport Signatures)

In this section we give a very simple and elegant construction of a one-time-secure digital signature scheme, due to Lamport.[2] The construction is a straightforward generalization of the signature scheme constructed above: Each message is signed bit-by-bit, where each bit is signed using a fresh and independently generated secret key.

Before giving the construction, we define one-time security for digital-signature schemes. This signature scheme is not generally useful on its own, but is useful as a building block.

[2] Leslie Lamport. *Constructing Digital Signatures from a One Way Function*. Tech. rep. Oct. 1979.

Definition 4.1 (Digital signatures: One-Time Security)**.** A digital-signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ over message space $\mathcal{M}$ is *one-time secure* if all efficient adversaries win the following game with negligible probability:

- The challenger generates $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)$ and sends vk to the adversary.
- The adversary sends the challenger *single* message $m \in \mathcal{M}$.
- The challenger responds with $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$.
- The adversary outputs (m^*, σ^*).
- The adversary wins the game if $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m^* \neq m$.

Lamport signatures. We now construct a one-time secure signature scheme for messages in $\{0,1\}^n$, for some fixed message length $n \in \mathbb{N}$. To do this, we will define the following algorithms, which make use of a one-way function $f \colon \mathcal{X} \to \mathcal{Y}$:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{vk})$. Choose $2n$ random elements from $\mathcal{X}$, the domain of the one-way function f. Arrange these values in to a $2 \times n$ matrix, which forms the secret signing key sk. The public verification key just consists of the $2n$ images of these values under the one-way function f:

$$\mathsf{sk} \leftarrow \begin{pmatrix} x_{10} & \dots & x_{n0} \\ x_{11} & \dots & x_{n1} \end{pmatrix}, \quad \mathsf{vk} \leftarrow \begin{pmatrix} f(x_{10}) & \dots & f(x_{n0}) \\ f(x_{11}) & \dots & f(x_{n1}) \end{pmatrix}.$$

- $\mathsf{Sign}(\mathsf{sk}, m) \to \sigma$ outputs $(x_{1m_1}, \dots x_{nm_n})$, where $m_1 \dots m_n$ are the individual bits of the length-n message $m \in \{0,1\}^n$.

- $\mathsf{Ver}(\mathsf{vk}, m, \sigma) \to \{0,1\}$ parses the the message into bits $m = m_1 \dots m_n \in \{0,1\}^n$ and the signature σ into its individual symbols $\sigma = (x_1^*, \dots x_n^*) \in \mathcal{X}^n$. The signing routine accepts if, for all $i \in \{1, \dots, n\}$:

$$f(x_i^*) = \mathsf{vk}_{i,m_i}. \tag{1}$$

  In other words, the routine accepts if applying the one-way function f to each symbol of the signature matches the corresponding value in the verification key. (Otherwise, the signing routine rejects.)

This signature scheme has relatively large keys: the verification key, in particular consists of $2n$ values, where each is of length $\Omega(\lambda)$ bits. So the total length is roughly $2n\lambda$ bits—much longer than the n-bit message being signed.

In addition, notice that an adversary who sees signatures on even two messages can forge signatures on messages of its choice. In particular:

- The adversary first asks for a signature on the message $m_0 = 0^n$. It receives $\sigma_0 = (x_{10}, \dots, x_{n0})$.
- The adversary then then asks for a signature on the message $m_1 = 1^n$. It receives $\sigma_1 = (x_{11}, \dots, x_{n1})$.

- At this point, the adversary has the entire secret signing key!

However, we will show that this scheme is indeed one-time secure.

Claim. *The Lamport signature scheme is one-time secure under the assumption that f is a one-way function.*

Remember that if $\mathsf{P} = \mathsf{NP}$, one-way functions, and also digital signature schemes, do not exist. So any proof of security of a digital-signature scheme will require some sort of cryptographic assumption.

In cryptography, we generally prove these security claims by *reduction*: we will show that if there exists an efficient adversary $\mathcal{A}$ that breaks the security of our scheme, then we can construct an efficient adversary $\mathcal{B}$ that breaks one of our assumptions. If we do this, we have reached a contradiction to one of our assumptions, so the first adversary cannot exist.

Proof of Claim. Suppose there exists an adversary $\mathcal{A}$ that wins the one-time-security game of Definition 4.1 with non-negligible probability ϵ. That is, the adversary can produces (m^*, σ^*) such that $\mathsf{Ver}(\mathsf{vk}, m^*, \sigma^*) = 1$ and $m \neq m^*$ given only $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$. We can then construct an adversary $\mathcal{B}$ that can use $\mathcal{A}$ to invert the one-way function.

Lamport's construction shows that if one-way functions exist, then so do digital signatures. Can you show that if digital signatures exist, then so do one-way functions?

In particular, our adversary $\mathcal{B}$ will use algorithm $\mathcal{A}$ as a subroutine to invert the one-way function. We will show that if $\mathcal{A}$ wins in the one-time signature security game often, then algorithm $\mathcal{B}$ will invert the one-way function often, which is a contradiction.

Assume our one-way function is of the form $f\colon \mathcal{X} \to \mathcal{Y}$ and that the Lamport signature scheme is on n-bit messages. The one-way-function adversary $\mathcal{B}$ operates as follows:

- The adversary $\mathcal{B}$ is given a point $y \in \mathcal{Y}$ and its task is to produce a preimage of y under f.
- The adversary $\mathcal{B}$ generates a signing keypair as follows:
  - It runs the key-generation algorithm for the Lamport signature scheme $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}()$.
  - The adversary chooses a random value $i^* \xleftarrow{R} \{1, \ldots, n\}$ and a random bit $\beta^* \xleftarrow{R} \{0, 1\}$.
  - The adversary sets $\mathsf{vk}_{i^*,\beta^*} \leftarrow y$. That is, it inserts the one-way-function point it must invert into a random location in the verification key.
- The adversary then sends the verification key vk to the Lamport-signature adversary $\mathcal{A}$.
- The adversary $\mathcal{A}$ asks for the signature on a message $m = m_1 m_2 \ldots m_n \in \{0, 1\}^n$.
- If $m_{i^*} = \beta^*$, then algorithm $\mathcal{B}$ cannot produce a valid signature on the message m and it outputs FAIL.
- Otherwise, the algorithm $\mathcal{B}$ returns the signature $\sigma = (\mathsf{sk}_{1,m_1}, \ldots, \mathsf{sk}_{n,m_n}) \in \mathcal{X}^n$ to algorithm $\mathcal{A}$.

- Algorithm $\mathcal{A}$ then produces a forged message-signature pair (m^*, σ^*), where $m \neq m^*$.
- Algorithm $\mathcal{B}$ parses $m^* = m_1^* \dots m_n^* \in \{0,1\}^n$ and $\sigma^* = \sigma_1^* \dots \sigma_n^* \in \mathcal{X}^n$. Then:
  - If $m_{i^*} = m_i$, algorithm $\mathcal{B}$ outputs FAIL.
  - Otherwise, algorithm $\mathcal{B}$ outputs $x \leftarrow \sigma_{i^*}^* \in \mathcal{X}$.

First, notice that whenever (m^*, σ^*) is a valid message-signature pair and whenever algorithm $\mathcal{B}$ does not output FAIL, algorithm $\mathcal{B}$ outputs a preimage $x \in \mathcal{X}$ of point $y \in \mathcal{Y}$ under the one-way function f. That is because, by the verification relation (1) for Lamport signatures,

$$f(x) = f(\sigma_{i^*}^*) = \mathsf{vk}_{i^*, m_{i^*}^*} = \mathsf{vk}_{i^*, 1-m_i} = \mathsf{vk}_{i^*, \beta^*} = y.$$

Now, we must show that algorithm $\mathcal{B}$ does not output FAIL too often. Since algorithm $\mathcal{B}$ chooses the values i^* and β^* at random, and since the adversary $\mathcal{A}$ behavior is *independent* of these values, we can say:

- the probability of the first failure event is $1/2$, since there are two possible choices of m_{i^*} and only one of these is bad, and
- the probability of the second failure event is at most $1/n$, since m and m^* must differ in at least one of n bits, and there is a $1/n$ probability that this differing bit is at index i^*.

The events that $\mathcal{A}$ breaks the signature scheme and that either of these failures occur are all *independent*. Then if $\mathcal{A}$ breaks the one-way function with probability ϵ, our one-way-function adversary $\mathcal{B}$ inverts the one-way function with probability

$$\epsilon_{\text{one-way}} = \epsilon \cdot \frac{1}{2} \cdot \frac{1}{n}.$$

The probability of either bad is at most $1/2 + 1/n$, by the union bound. Therefore if algorithm $\mathcal{A}$ breaks one-time security of Lamport's scheme with probability ϵ, If ϵ is non-negligible, then $\epsilon_{\text{one-way}} = \epsilon/2n$ is also non-negligible, and we have a contradiction. □

5 *A one-time signature scheme for arbitrary-length messages*

In the Lamport signature scheme (Section 4), the length of the keys scales with the size of the message being signed. To adapt our scheme from above into a scheme that works on arbitrary-length messages without the key growing arbitrarily large, we will use a strategy called *hash-and-sign*. In essence, the signing algorithm will pass the message through a hash function to generate a fixed-size

Essentially all signature schemes used in practice use this hash-and-sign construction.

digest before applying a signature scheme that works only on fixed-length messages.

This paradigm is called "hash and sign," and is very common. In practice, hashing is computationally cheap operation while signing turns out to be computationally relatively expensive. So it is common to hash a message before signing it in order to reduce the size of the message that must be signed.

The following claim gives the general construction:

Claim (Hash-and-sign paradigm)**.** *Given a collision-resistant hash function* $h : \{0,1\}^* \to \{0,1\}^n$ *and a signature scheme* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *for message space* $\mathcal{M} = \{0,1\}^n$ *(such as the one in Section 4), there exists a signature scheme* $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ *for* $\mathcal{M}' = \{0,1\}^*$ *as follows:*

- $\mathsf{Gen}' := \mathsf{Gen}$. *The key-generation algorithm is unchanged.*
- $\mathsf{Sign}'(\mathsf{sk}, m) := \mathsf{Sign}(sk, h(m))$. *We hash the message using the hash function h before passing the hashed message to the original signing function.*
- $\mathsf{Ver}'(\mathsf{vk}, m, \sigma) := \mathsf{Ver}(\mathsf{vk}, h(m), \sigma)$. *We use the original* Ver *to check that the tag matches* hash *of the original message.*

Security Intuition. Suppose that there exists an efficiency adversary that breaks $(\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$. In particular, given $((m_1, \sigma_1, \ldots, (m_t, \sigma_t))$, the adversary is able to construct a valid message-signature pair (m^*, σ^*) such that $m^* \notin \{m_1, \ldots, m_t\}$. There are then two cases:

1. $h(m^*) \in \{h(m_1), \ldots, h(m_t)\}$. If this is the case, there is some $i \in [t]$ such that $h(m^*) = h(m_i)$. However, h is collision-resistant as in the definition, so this is a contradiction!
2. $h(m^*) \notin \{h(m_1), \ldots, h(m_t)\}$. Since the message that we pass to the underlying signature scheme is $h(m)$, this means that the adversary has found a valid signature for $h(m)$ under the original scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ after seeing only signatures of $h(m_1) \neq h(m), \ldots, h(m_t) \neq h(m_t)$. This breaks the security of the underlying signature scheme, which is a contradiction!

Application to Lamport. We can apply the hash-and-sign paradigm to the Lamport signature scheme from Section 4. We can fix our input to the Lamport scheme at, for example, 256 bits, and then run messages through a hash function that outputs 256 bits before passing them to the Lamport scheme. This gives a *one-time-secure* signature scheme for messages of arbitrary length.

Security implications of hash and sign. In practice, hash-and-sign can actually *increase* the security of our signature scheme, in a certain sense. As shown in case 2 above, it is absolutely crucial that the hash function used is collision-resistant: if not, an adversary can find messages that cause collisions, and then a signature for one message will also be a valid signature for the other. However, in practice we often think of hash functions like SHA256 as behaving like *random oracles*. That is, for a hash function $h\colon \{0,1\}^* \to \{0,1\}^\lambda$ and a string $x \in \{0,1\}^*$ we think of the value $h(x)$ as being an independently sampled and uniformly random value from the co-domain of the hash function, $\{0,1\}^\lambda$. (Of course, a real-world hash function is *never actually* a random oracle. A random oracle from $h\colon \{0,1\}^* \to \{0,1\}^\lambda$ would take infinitely many bits to describe, while real-world hash functions have finite size (and polynomial-size descriptions).)

Recall that we faced a similar problem in our MAC construction. Why not use hash and MAC there? The reason is that we used AES as our PRF, which takes input of $\{0,1\}^{128}$. As explained by the birthday paradox, it is possible to find a collision in an output space of size 2^{128} in only time 2^{64}! This does not provide sufficient security for practical use, as it would be quite practical to find collisions. If we had a version of AES that outputted 256 bits, we could indeed apply hash and sign.

Another reason to not use hash and MAC is that MACs can be faster to compute than collision-resistant hash functions.

Recall that the standard security definition for digital signatures (Definition 1.3) allows the attacker to request signatures on messages of its choice. If we pass a message through a hash function before signing it using an underlying signature scheme scheme, however, we effectively randomize the message—the adversary can no longer control the input to the underlying signature scheme. This allows us to define another meaningful definition of security:

Definition 5.1 (Digital signatures: security against random message attacks)**.** Any efficient adversary given the public verification key and a list of random message-signature pairs $((m_1,\sigma_1),\ldots,(m_t,\sigma_t))$ cannot generate a forged message-signature pair (m^*,σ^*) such that $\mathsf{Ver}(\mathsf{vk},m^*,\sigma^*) = 1$ and $m^* \notin \{m_1,\ldots,m_t\}$.

Note that this definition is *not* good enough on its own—an adversary often does have the ability to generate signatures for messages of his choice. However, paired with a hash function modeled as a random oracle, this definition becomes very useful—if the inputs are passed through the hash function before they are passed to the signature scheme, they become effectively random. We can even relax the definition further without losing practicality: by the same logic, with hash function in front of the signature scheme, what the adversary needs to sign is really not a message of their choise, but is the *hash* of a message of their choice—effectively a random value.

Definition 5.2 (Digital signatures: random security against random message attacks)**.** Any efficient adversary given vk and a list of random message-signature pairs $((m_1,\sigma_1),\ldots,(m_t,\sigma_t))$ and random $m^* \notin \{m_1,\ldots,m_t\}$ cannot generate σ^* such that $\mathsf{Ver}(\mathsf{vk},m^*,\sigma^*) = 1$.

It is possible to formally argue that given a signature scheme satisfying Definition 5.2 and a random oracle, we can construct a

scheme satisfying existential unforgeability under chosen message attacks.

6 *From one-time security to many-time security*

After applying the hash-and-sign strategy above to our Lamport scheme, we have a signature scheme that is one-time secure for messages of arbitrary length. In order for the scheme to be useful and satisfy our security definition, we need to be able to sign polynomially many messages with a single key pair. To do this, we will use a construction very similar to the Merkle tree construction we have seen before.

Informally, we will build up a binary tree of Lamport keys of depth 256. The signing key in each of the 2^{256} leaf nodes will be used to actually sign messages; we will use a random leaf node to sign a message, so the fact that there are 2^{256} of them means that the probability of accidentally choosing the same leaf twice is negligible (2^{-128}). The signing key in an intermediate nodes (and in the root) will be used to sign the (public) verification keys of the two corresponding child nodes in the tree. Fig. 1 shows a sketch of this tree.

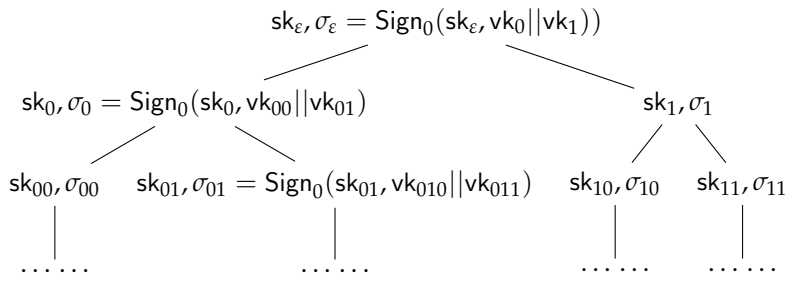

Figure 1: Sketch of the tree of Lamport signature keys used for the many-time secure signature construction.

Importantly, every signing key in the tree will be used to sign only one message ever: the key at non-leaf nodes will only ever sign the pair of its children, and the key at leaf nodes will only ever be used to sign a single message.

The signature, in this scheme, will consist of the signatures and verification keys along the path from the root to the chosen leaf node. The signature will also include information about which child node was chosen.

This tree, of course, is impractically large, but we can solve that problem by lazily constructing it using a pseudorandom function. That is, instead of actually building up all of the leaves of the tree, we will build up the tree (i.e., generate the signing keys, verification keys, and signatures) on-demand, and furthermore, we will build it in a

deterministic way using the pseudorandom function, so that we don't have to remember what parts we might have already computed in the past.

To make the construction more precise, we will assume that we are given:

- a pseudorandom function f with keyspace $\mathcal{K}$, and
- a one-time secure signature scheme $(\mathsf{Gen}_0, \mathsf{Sign}_0, \mathsf{Ver}_0)$.

We will need the ability to run the (non-deterministic) Gen_0 algorithm on specific randomness, so as to make it deterministic. For a PRF key $k \in \mathcal{K}$ and string s, we will write $\mathsf{Gen}_0^{F(k,s)}()$ to indicate the process of running the key-generation algorithm Gen_0 using randomness derived from the output of the PRF $F(k,s)$. We will assume the use of SHA256 (with a 256-bit digest length) as a collision-resistant hash function. Using these building blocks, we will construct a many-time secure signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ for arbitrary-length messages (i.e., on message space $\{0,1\}^*$), where all of the algorithms are efficient (poly$(\cdot)$ running time).

Our construction is as follows:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{vk})$:
  - Sample a fresh PRF key $k \xleftarrow{R} \mathcal{K}$.
  - Set $(\mathsf{sk}_\epsilon, \mathsf{vk}_\epsilon) \leftarrow \mathsf{Gen}_0^{F(k,\text{“”})}()$.
  - Output $(\mathsf{sk}, \mathsf{vk}) \leftarrow (k, \mathsf{vk}_\epsilon)$.
- $\mathsf{Sign}_t(k, m) \to \sigma$:
  - Choose a random 256-bit value $r = (r_1 \dots r_{256}) \in \{0,1\}^{256}$).
  - Compute $(\mathsf{sk}_r, \mathsf{vk}_r) \leftarrow \mathsf{Gen}_0^{F(k,r)}()$
  - Compute $\sigma_r \leftarrow \mathsf{Sign}_0(\mathsf{sk}_r, \mathrm{SHA256}(m))$.
  - Compute σ_ε, vk_0, vk_1, σ_{r_1}, $\mathsf{vk}_{r_1 0}$, $\mathsf{vk}_{r_1 1}$, $\sigma_{r_1 r_2}$, …, $\sigma_{r_1 r_2 \dots r_{255}}$, $\mathsf{vk}_{r_1 r_2 \dots r_{255} 0}$, $\mathsf{vk}_{r_1 r_2 \dots r_{255} 1}$ as shown in Fig. 1.
  - Output $\sigma \leftarrow (r, \sigma_\varepsilon, \mathsf{vk}_0, \mathsf{vk}_1, \sigma_{r_1}, \mathsf{vk}_{r_1 0}, \mathsf{vk}_{r_1 1}, \dots, \sigma_r)$.
- $\mathsf{Ver}_t(\mathsf{vk}_\epsilon, m, \sigma) \to \{0,1\}$:
  - Parse $(r, \sigma_\varepsilon, \mathsf{vk}_0, \mathsf{vk}_1, \sigma_{r_1}, \mathsf{vk}_{r_1 0}, \mathsf{vk}_{r_1 1}, \dots, \sigma_r) \leftarrow \sigma$.
  - Output "1" if and only if
    - $\mathsf{Ver}_0(\mathsf{vk}_r, \sigma_r, \mathrm{SHA256}(m)) = 1$ and
    - $\mathsf{Ver}_0(\mathsf{vk}_x, \sigma_x, \mathsf{vk}_{x0} || \mathsf{vk}_{x1}) = 1$ for every prefix x of r (from ε to $r_1 r_2 \dots r_{255}$).

7 *Choosing Signature Schemes*

The signature scheme we presented in Section 6 is not particularly efficient in terms of signature size.

Algorithm	vk size	σ size	signatures/sec	verifications/sec
SPHINCS+-128	32 B	8000 B	5	750
RSA 2048	256 B	256 B	2,000	50,000
ECDSA256	32 B	64 B	42,000	14,000

Table 1: Statistics about various signature schemes used in practice

Many deployed systems today use the ECDSA256 signature scheme. Legacy application still use RSA signatures, though because of their relatively large public-key and signature sizes, few new applications use these schemes.

Hashing is much, much faster than signing—the commonly used SHA256 hashing algorithm can compute around 10,000,000 hashes per second. This is one reason the hash-and-sign paradigm is so useful.

References

Diffie, Whitfield and Martin E Hellman. "New Directions in Cryptography". In: *Transactions on Information Theory* 22.6 (1976).

Lamport, Leslie. *Constructing Digital Signatures from a One Way Function*. Tech. rep. Oct. 1979.