Fragments of a chapter on Signature Schemes

(Extracts from Foundations of Cryptography – in preparation)

Oded Goldreich

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.

July 25, 2000

Contents

Chapter 6

Digital Signatures and Message Authentication

Message authentication and (digital) signatures were the first tasks that joined encryption to form modern cryptography. Both message authentication and digital signatures are concerned with the "authenticity" of data, and the difference between them is analogous to the difference between private-key and public-key encryption schemes.

In this chapter, we define message authentication and digital signatures, and the security notions associated to them. We show how to construct message authentication schemes using pseudorandom functions, and how to construct signature schemes using one-way permutations. We stress that the latter construction employ one-way permutations that do not necessarily have a trapdoor.

6.1 Definitional Issues

Message authetication and signature schemes are supposed to enable reliable transmission of data between parties. Loosely speaking, the receiver wishes to be guaranteed that the data received was actually sent by the sender, rather than modified (or even totally injected) by a third party. The difference between message authetication and signature schemes lies in the identity of the receiver and the level of trust that the sender has in it. Typically, message authentication schemes are employed in cases where the receiver is predetermined (at the time of message transmission), whereas signature shemes allow verification of the authenticity of the data by anybody. In other words, signature schemes allow for universal verification, whereas message authentication schemes may only allow predetermine parties to verify the authenticity of the data. In both cases, the authentication process consists of two main processes: the generation of message-authentication tags or signatures by the allenged sender, and the verification of such tags or signatures by the receiver. As in case of encryption schemes, there is a third (implicit) process that allows the sender to generate a tagging/signing key, along with a verification key. (The possession of the former key constitutes the sender's advantage over the adversary; see analogous discussion in the previous chapter.)

6.1.1 Message authentication versus signature schemes

The difference between message-authentication and signature schemes amounts to the question of whether the adversary knows the verification key. In message-authentication schemes, the verification key is only given to a set of predetermined receivers that are all trusted not to abuse this knowledge; that is, in such schemes it is postulated that the verification key is not (a-priori) known to the adversary. On the other hand, in signature schemes, the aim is universal verification and so the verification key is public, and hence known also to the adversary.

Summary and terminology: Message authentication and signature schemes differ in the question of whether the verification key is secret (i.e., unknown to the adversary) or public (and also known to the adversary). Thus, in a sense these are private-key and public-key versions of a task which lacks a good name (since both authentication and signatures are already taken by one of the versions). Still, seeking a uniform terminology, we shall sometimes refer to message authentication schemes (also known as *message authentication codes* (MAC)) as to private-key signature schemes. Analogously, we shall sometimes refer to signature schemes as to public-key signature schemes.

6.1.2 Basic mechanism

We start by defining the basic *mechanism of message-authentication and signature schemes*. Recall that there are private-key and public-key versions, but the difference between the two version is only reflected in the definition of security. In contrast, the definition of the basic mechanism says nothing about the security of the scheme (which is the subject of the next section), and thus is the same for both the private-key and public-key versions. In both cases, the scheme consists of three efficient algorithms: *key generation*, *signing* (or *authenticating*) and *verification*. The basic requirement is that signatures that are produced by the signing algorithm be accepted as valid by the verification algorithm, when fed a verification-key corresponding to the signing-key used by the signing algorithm.

Definition 6.1.1 (signature scheme): *A* signature scheme *is a triple,* (G, S, V)*, of probabilistic polynomial-time algorithms satisfying the following two conditions*

1. *On input* 1^n*, algorithm* G *(called the* key generator*) outputs a pair of bit strings.*

2. *For every pair* (s, v) *in the range of* $G(1^n)$*, and for every* $\alpha \in \{0,1\}^*$*, algorithms* S (signing) *and* D (verification) *satisfy*

$$\mathsf{Pr}[V(v, \alpha, S(s, \alpha)) = 1] = 1$$

where the probability is taken over the internal coin tosses of algorithms S *and* V*.*

The integer n *serves as the* security parameter *of the scheme. Each* (s, v) *in the range of* $G(1^n)$ *consitutes a pair of corresponding* signing/verification keys*. The string* $S(s, \alpha)$ *is a* signature to the document $\alpha \in \{0,1\}^*$ *using the signing key* s*.*

We stress that Definition 6.1.1 says nothing about security, and so trivial (insecure) algorithms may satisfy it (e.g., $S(s, \alpha) \stackrel{\text{def}}{=} 0$ and $V(v, \alpha, \beta) \stackrel{\text{def}}{=} 1$). Furthermore, Definition 6.1.1 does not distinguish private-key signature schemes from public-key ones. The difference between the two types is introduced in the security definitions: In a public-key scheme the "forging algorithm" gets the verification key (i.e., v) as an additional input (and thus $v \neq s$ follows); while in private-key schemes v is not given to the "forging algorithm" (and thus one may assume, without loss of generality, that $v = s$).

Notation: In the rest of this book, we write $S_s(\alpha)$ instead of $S(s, \alpha)$ and $V_v(\alpha, \beta)$ instead of $V(v, \alpha, \beta)$. Also, we let $G_1(1^n)$ (resp., $G_2(1^n)$) denote the first (resp., second) element in the pair $G(1^n)$. That is, $G(1^n) = (G_1(1^n), G_2(1^n))$. Without loss of generality, we may assume that $|G_1(1^n)|$ and $|G_2(1^n)|$ are polynomially related to n, and that each of these integers can be efficiently computed from the other.

Comments: The above definition may be relaxed in several ways without significantly harming its usefulness. For example, we may relax Condition (2) and allow a negligible verification error (e.g., $\mathsf{Pr}[V(v, \alpha, S(s, \alpha)) \neq 1] = 2^{-n}$). Alternatively, one may postulate that Condition (2) holds for all but a negligible measure of the key-pairs generated by $G(1^n)$. At least one of these relaxations is essential for many suggestions of (public-key) signature schemes.

Another relaxation consists of restricting the domain of possible documents. However, unlike the situation with respect to encryption schemes, such a restriction is non-trivial in the current context, and is discussed at length in Section 6.2.

6.1.3 Attacks and security

We consider very powerful attacks on the signature scheme as well as a very liberal notion of breaking it. Specifically, the attacker is allowed to obtain signatures to any document of its choice. One may argue that in many applications such a general attack is not possible (as documents to be signed must have a specific format). Yet, our view is that it is impossible to define a general (i.e., application-independent) notion of admissible documents, and thus a general/robust definition of an attack seems to have to be formulated as suggested here. (Note that at worst, our approach is overly cautious.) Likewise, the adversary is said to be successful if it can produce a valid signature to ANY document for which it has not asked for a signature during its attack. Again, this defines the ability to form signatures to possibly "nonsensical" documents as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" documents (so that only forging signatures to them will be consider a breaking of the scheme). The above discussion leads to the following definition.

Definition 6.1.2 (unforgeable signatures): *We start with a slightly imformal outline.*

- *A* chosen message attack *is a process that can obtain signatures to strings of its choice, relative to some fixed signing-key that is generated by G. We distinguish two cases.*

  The private-key case: *Here the attacker is given 1^n as input, and the signatures are produced relative to s, where $(s,v) \leftarrow G(1^n)$.*

  The public-key case: *Here the attacker is given v as input, and the signatures are produced relative to s, where $(s,v) \leftarrow G(1^n)$.*

- *Such an attack is said to* succeeds (in existential forgery) *if it outputs a valid signature to a string for which it has* NOT *requested a signature during the attack. That is, the attack is successful if it outputs a pair (α,β) so that α is different from all strings for which a signature has been required during the attack, and $\Pr[V_v(\alpha,\beta)=1] \geq \frac{1}{2}$, where v is as above.*[1]

- *A signature scheme is* secure (or unforgeable) *if every probabilistic polynomial-time chosen message attack succeeds with at most negligible probability.*

Formally, a chosen message attack is modelled by a probabilistic oracle machine. Let M be such a machine. We denote by $Q_M^O(x)$ the set of queries made by M on input x and access to oracle O, and let $M_1^O(x)$ denote the first string in the pair of strings output by M on input x and access to oracle O.

The private-key case: *A* private-key signature scheme *is* secure *if for every probabilistic polynomial-time oracle machine M, every polynomial p and all sufficiently large n, it holds that*

$$\Pr\left[V_{G_2(1^n)}(M^{S_{G_1(1^n)}}(1^n))=1 \ \&\ M_1^{S_{G_1(1^n)}}(1^n) \notin Q_M^{S_{G_1(1^n)}}(1^n)\right] < \frac{1}{p(n)}$$

[1] The threshold of $1/2$ used above is quite arbitrary. The definition is essentially robust under the replacement of $1/2$ by either $1/\text{poly}(n)$ or $1-2^{-\text{poly}(n)}$, by amplification of the verification algorithm. For example, given V as above, one may consider V' that applies V to the tested pair for a linear number of times and accepting if and only if V has accepted in all tries.

where the probability is taken over the coin tosses of algorithms G, S and V as well as over the coin tosses of machine M.

The public-key case: *A* public-key signature scheme *is* secure *if for every probabilistic polynomial-time oracle machine M, every polynomial p and all sufficiently large n, it holds that*

$$\Pr\left[\begin{array}{c} V_{G_2(1^n)}(M^{S_{G_1(1^n)}}(G_2(1^n)))=1 \\ \textit{and} \\ M_1^{S_{G_1(1^n)}}(G_2(1^n)) \notin Q_M^{S_{G_1(1^n)}}(G_2(1^n)) \end{array}\right] < \frac{1}{p(n)}$$

where the probability is taken over the coin tosses of algorithms G, S and V as well as over the coin tosses of machine M.

The definition refers to the following experiment. First a pair of keys, (s, v), is generated by invoking $G(1^n)$, and is fixed for the rest of the discussion. Next, an attacker is given oracle access to S_s, where the latter may be a probabilistic oracle rather than a standard deterministic one (e.g., if queried twice for the same value then the signing oracle may answer in different ways). Finally, the attacker outputs a pair of strings (α, β), and is deemed successful if and only if the following two conditions hold:

1. The string α is different than all queries (i.e., requests for signatures) made by the attacker; that is, $M_1^{S_s}(x) \notin Q_M^{S_s}(x)$, where $x = 1^n$ or $x = v$ depending on whether we are in the private-key or public-key case. (We stress that both $M_1^{S_s}(x)$ and $Q_M^{S_s}(x)$ are random variables that are defined based on the *same* random execution of M on input x and oracle access to S_s.)

2. The pair (α, β) corresponds to a valid document-signature pair relative to the verification key v. In case V is deterministic (which is typically the case) this means that $V(\alpha, \beta) = 1$. The same applies also in case V is probabilistic, and when viewing $V(\alpha, \beta) = 1$ as a random variable. (Alternatively, in the latter case, a condition such as $\Pr[V(\alpha, \beta) = 1] \geq 1/2$ may replace the condition $V(\alpha, \beta) = 1$.)

Clearly any signature scheme that is secure in the public-key model is also secure in the private-key model. The converse is not true: consider, for example, the private-key scheme presented in Construction 6.3.1 (below).

Failure of some popular schemes: We stress that *plain* RSA (alike plain versions of Rabin's scheme and DSS) is not secure under the above definition. However, variants of these signature schemes may be secure if the message is "randomized" before RSA (or the other schemes) is applied. In general, the randomization paradigm (see Section 5.3) will play a pivotal role in this chapter too.

6.2 Signing fixed length documents versus abritrary ones

Restricted types of (public-key and private-key) signature schemes play an important role in our exposition. The first restriction we consider is the one of schemes capable of signing only documents of certain predetermined length. The effect of this restriction is more dramatic here (in the context of signature schemes) than it was in the context of encryption schemes; compare Section 5.3.2. Still, we shall show (see Theorem 6.2.2 below) that if the length restriction is not too low then the full power of signature schemes can be regained.

Definition 6.2.1 (signature scheme for fixed length documents): *Let* $\ell : \mathbb{N} \to \mathbb{N}$*. An* ℓ-restricted signature scheme *is a triple,* (G, S, V)*, of probabilistic polynomial-time algorithms satisfying the following two conditions*

1. *As in Definition 6.1.1, on input* 1^n*, algorithm* G *outputs a pair of bit strings.*

2. *Analogously to Definition 6.1.1, for every* n *and every pair* (s, v) *in the range of* $G(1^n)$*, and for every* $\alpha \in \{0,1\}^{\ell(n)}$*, algorithms* S *and* D *satisfy* $\mathsf{Pr}[V(v, \alpha, S(s, \alpha)) = 1] = 1$.

Such a scheme is called secure (in the private-key or public-key model) *if the* (corresponding) *requirements of Definition 6.1.2 hold when restricted to attackers that only make queries of length* $\ell(n)$ *and output a pair* (α, β) *with* $|\alpha| = \ell(n)$.

We comment that ℓ-restricted private-key signature schemes for $\ell(n) = O(\log n)$ are trivial (since the signing and verification keys may contain a table look-up associating a secret with each of the $2^{\ell(n)} = \mathrm{poly}(n)$ possible documents; see Exercise 1). (In contrast, this triviality does not arise in the context of encryption schemes; compare Section 5.3.2.) On the other hand, ℓ-restricted signature schemes for super-logarithmic ℓ (e.g., $\ell(n) = n$ or even $\ell(n) = \log_2^2 n$ will do) are as powerful as ordinary signature schemes:

Theorem 6.2.2 *Suppose that* ℓ *is a super-logarithmically growing function Then, given an* ℓ*-restricted signature scheme that is secure in the private-key* (resp., public-key) *model, one can construct a full-fledged signature scheme that is secure in the same model.*

The theorem can be proved in two different ways, corresponding to two methods of converting an ℓ-restricted signature scheme into a full-fledged one. The first method consists of parsing the original document into blocks (with proper binding between blocks!), and applying the ℓ-restricted scheme to each block. The second method consists of hashing the document into an $\ell(n)$-bit long value (via an adequate hashing scheme!), and applying the restricted scheme to the resulting value. Although the theorem can be proved using each of the two methods, we only prove the theorem using the first method. The second method is presented because it is actually much more important (as we may see in Section 6.4).

6.2.1 First method: signing (augmented) document blocks

Let ℓ and (G,S,V) be as in Theorem 6.2.2. We construct a general signature scheme, (G',S',V'), with $G'=G$, by considering documents as sequences of strings each of length $\ell'(n)=\ell(n)/O(1)$. That is, associate $\alpha=\alpha_1\cdots\alpha_t$ with the sequence $(\alpha_1,...,\alpha_t)$, where each α_i has length $\ell'(n)$.

To motivate the following construction, consider the following simpler schemes aimed at producing secure signatures for sequences of $\ell'(n)$-bit long strings. The simplest idea is to *just sign each of the strings in the sequence*. That is, the signature to the sequence $(\alpha_1,...,\alpha_t)$, is a sequence of β_i's each being a signature (w.r.t the length-restricted scheme) to the corresponding α_i. This will not do since an adversary, given a single signature (β_1,β_2) to the sequence (α_1,α_2) with $\alpha_1\neq\alpha_2$, can present (β_2,β_1) as a signature to (α_2,α_1). So how about signing the sequence $(\alpha_1,...,\alpha_t)$ by applying the restricted scheme to each pair (i,α_i), so to foil the above attack? This will not do either, since an adversary, given a signature to the sequence (α_1,α_2) can easily present a signature to the sequence (α_1). So we need to include in each $\ell(n)$-bit string also the total number of α_i's in the sequence. But even this is not enough, since an adversary given signatures to the sequences (α_1,α_2) and (α_1',α_2'), with $\alpha_1\neq\alpha_1'$ and $\alpha_2\neq\alpha_2'$, can easily generate a signature to (α_1,α_2'). Thus, we have to prevent the forming of new sequences of basic signatures by combination of elements from different signature sequences. This can be done by associating (say at random) an identificator with each sequence and incorporating this identificator in each $\ell(n)$-bit string to which the restricted scheme is applied. This yields the following signature scheme:

Construction 6.2.3 (signing augmented blocks): *Let ℓ and (G,S,V) be as in Theorem 6.2.2. We construct a general signature scheme, (G',S',V'), with $G'=G$, by considering documents as sequences of strings. We construct S' and V' as follows, using $G'=G$ and $\ell'(n)=\ell(n)/4$.*

signing with S': *On input a signing-key $s\in G_1(1^n)$ and a document $\alpha\in\{0,1\}^*$, algorithm S' first parses α into $\alpha_1,...,\alpha_t$ so that α is uniquely reconstructed from the α_i's and each α_i is an $\ell'(n)$-bit long string.*[2] *Next, S' uniformly selects $r\in\{0,1\}^{\ell'(n)}$. For $i=1,...,t$, algorithm S' computes*

$$\beta_i \leftarrow S_s(r,t,i,\alpha_i)$$

That is, β_i is a signature to the statement "α_i is the i^{th} block, out of t blocks, in a sequence associate with identificator r". *Finally, S' outputs as signature the sequence*

$$(r,t,\beta_1,....,\beta_t)$$

[2] For example, we may require that $\alpha\cdot 10^j=\alpha_1\cdots\alpha_t$ and $j<\ell'(n)$.

verification with V': *On input a verifying-key* $v \in G_2(1^n)$, *a document* $\alpha \in \{0,1\}^*$, *and a sequence* $(r,t,\beta_1,....,\beta_t)$, *algorithm* V' *first parses* α *into* $\alpha_1,...,\alpha_{t'}$, *using the same parsing rule as* S'. *Algorithm* V' *accepts if and only if the following two conditions hold:*

1. $t' = t$, *where* t' *is obtained in the parsing of* α *and* t *is part of the alledge signature.*
2. *For* $i = 1,...,t$, *it holds that* $V_v((r,t,i,\alpha_i),\beta_i)$, *where* α_i *is obtained in the parsing of* α *and the rest are as in the corresponding parts of the alledge signature.*

Clearly, the triplet (G',S',V') satisfies Definition 6.1.1. We need to show that is also inherits the security of (G,S,V). That is,

Proposition 6.2.4 *Suppose that* (G,S,V) *is an* ℓ*-restricted signature scheme that is secure in the private-key* (resp., public-key) *model. Then* (G',S',V'), *as defined in Construction 6.2.3 is a full-fledged signature scheme that is secure in the private-key* (resp., public-key) *model.*

Theorem 6.2.2 follows immediately from Proposition 6.2.4.

Proof: The proof is by a reducibility argument. Given an adversary A' attacking the complex scheme (G',S',V'), we construct an adversary A that attacks the ℓ-restricted scheme, (G,S,V). In particular, the adversary A will have to use its oracle access in order to emulate the oracle S'_s for A'. This can be done in a straightforward manner; that is, algorithm A will act as S'_s does using the oracle S_s. (Specifically, A parses each query α' of A' into a corresponding sequence $(\alpha'_1,...,\alpha'_{t'})$, uniformly selects an identifier r', and obtains S_s signatures to (r',t',j,α'_j), for $j = 1,...,t'$.) When A' outputs a document-signature pair relative to the complex scheme (G',S',V'), algorithm A tries to use it in order to form a document-signature pair relative to the ℓ-restricted scheme, (G,S,V).

We stress that from the point of view of adversary A', the distribution of keys and oracle answers that A provides it with is exactly as in a real attack on (G',S',V'). This is a crucial point since we use the fact that events that occur in a real attack of A' on (G',S',V'), occur with the same probability in the emulation of (G',S',V') by A.

Assume that with (non-negligible) probability $\varepsilon'(n)$, the (probabilistic polynomial-time) algorithm A' succeeds in existentially forging relative to the complex scheme (G',S',V'). We consider the following cases regarding the forging event:

1. The identificator supplied in the forged signature is different from the random identificators supplied (by A) as part of the signatures given to A'. In this case, each ℓ-restricted signature supplied as part of the forged (complex) signature, yields existential forgery relative to the ℓ-restricted scheme.

Formally, let $\alpha^{(1)}, ..., \alpha^{(m)}$ be the sequence of queries made by A', and let $(r^{(1)}, t^{(1)}, \overline{\beta}^{(1)}), ..., (r^{(m)}, t^{(m)}, \overline{\beta}^{(m)})$ be the corresponding (complex) signatures supplied to A' by A (using S_s to form the $\overline{\beta}^{(i)}$'s). Let $(\alpha, (r, t, \beta_1, ..., \beta_t))$ be the output of A', and suppose that applying V'_v to it yields 1 (i.e., it is a valid document-signature pair for the complex scheme). It follows that each $\overline{\beta}^{(i)}$ consists of a sequence of S_s-signatures to $\ell(n)$-bit strings starting with $r^{(i)} \in \{0,1\}^{\ell(n)/4}$, and that the oracle S_s was invoked (by A) only on strings of this form. The case hypothesis states that $r \neq r^{(i)}$, for all i's. It follows that each of the β_j's is an S_s-signature to a string starting with $r \in \{0,1\}^{\ell(n)/4}$, and thus different from all queries made to the oracle S_s. Thus, each pair $((r, t, i, \alpha_i), \beta_i)$ is a valid document-signature pair (since $V'_v(\alpha, (r, t, \beta_1, ..., \beta_t)) = 1$ implies $V_v((r, t, i, \alpha_i), \beta_i) = 1$), with a document different than all queries made to S_s. This yields a forgery success relative to the ℓ-restricted scheme.

2. The identificator supplied in the forged signature equals the random identificator supplied (by A) as part of one of the signatures given to A'. (We stress that the latter signature is unique, and deal with the event in which it is not unique in the next case.)

   Formally, let $\alpha^{(1)}, ..., \alpha^{(m)}$ be the sequence of queries made by A', and let $(r^{(1)}, t^{(1)}, \overline{\beta}^{(1)}), ..., (r^{(m)}, t^{(m)}, \overline{\beta}^{(m)})$ be the corresponding (complex) signatures supplied to A' by A (using S_s to form the $\overline{\beta}^{(i)}$'s). Let $(\alpha, (r, t, \beta_1, ..., \beta_t))$ be the output of A', and suppose that applying V'_v to it yields 1 (i.e., it is a valid document-signature pair for the complex scheme). The hypothesis of the current case is that there exists a unique i so that $r = r^{(i)}$.

   We consider two subcases regarding the relation between t and $t^{(i)}$:

   - $t \neq t^{(i)}$. In this subcase, each ℓ-restricted signature supplied as part of the forged (complex) signature, yields existential forgery relative to the ℓ-restricted scheme. The argument is analogous to the one employed in the previous case. Specifically, here each of the β_j's is an S_s-signature to a string starting with (r, t), and thus different from all queries made to the oracle S_s (since these queries either start with $r^{(i')} \neq r$ or start with $(r^{(i)}, t^{(i)}) \neq (r, t)$). Thus, each pair $((r, t, j, \alpha_j), \beta_j)$ is a valid document-signature pair with a document differnt than all queries made to S_s.
   - $t = t^{(i)}$. In this case we use the hypothesis $\alpha \neq \alpha^{(i)}$, which implies that there exists a j so that $\alpha_j \neq \alpha_j^{(i)}$, where $\alpha_j^{(i)}$ is the j^{th} block in the parsing of $\alpha^{(i)}$. In this subcase, β_j (supplied as part of the forged complex-signature), yields existential forgery relative to the ℓ-restricted scheme. Specifically, we have $V_v((r, t, j, \alpha_j), \beta_j) = 1$, and in each query $(r^{(i')}, t^{(i')}, j', \alpha_{j'}^{(i')})$ made by A to S_s either $r^{(i')} \neq r$ (i.e., $i' \neq i$) or $j' \neq j$ or $\alpha_j \neq \alpha_j^{(i)}$. Thus, $((r, t, j, \alpha_j), \beta_j)$ is a valid

document-signature pair with a document differnt than all queries made to S_s.

3. The identificator supplied in the forged signature equals the random identificators supplied (by A) as part of at least two signatures given to A'. In particular, it follows that two signatures given to A use the same random identificator. The probability that this event occurs is at most

$$\binom{m}{2} \cdot 2^{-\ell'(n)} \; < \; m^2 \cdot 2^{-\ell(n)/4}$$

However, $m = \text{poly}(n)$ (since A' runs in polynomial-time), and $2^{-\ell(n)/4}$ is negligible (since ℓ is super-logarithmic). So this case occurs with negligible probability, and may be ignored.

Note that A can easily determine which of the cases occurs and act accordingly.[3] Thus, assuming that A' forges relative to the complex scheme with non-negligible probability $\varepsilon'(n)$, it follows that A forges relative to the complex scheme with non-negligible probability $\varepsilon(n) \geq \varepsilon'(n) - m^2 \cdot 2^{-\ell(n)/4}$, in contradiction to the proposition's hypothesis. ■

6.2.2 Second method: signing a hash value

Let ℓ and (G, S, V) be as in Theorem 6.2.2. The second method of constructing a general signature scheme out of (G, S, V) is based on the *hash then sign* paradigm. That is, first the document is hashed to an $\ell(n)$-bit long value, and then the ℓ-restricted scheme is applied to the hashed value. Thus, in addition to an ℓ-restricted scheme, this method employs an adequate hashing scheme. In particular, one way of implementing this method is based on "collision-free hashing" (defined next). An alternative implementation, based on "universal one-way hashing" is deferred to Section 6.4.3.

Loosely speaking, a *collision-free hashing scheme* consists of a collection of functions $\{h_s : \{0,1\}^* \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$ so that given s and x it is easy to compute $h_s(x)$, but given s it is hard to find $x \neq x'$ so that $h_s(x) = h_s(x')$.

Definition 6.2.5 (collision-free hashing functions): *Let $\ell : \mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^* \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is called* collision-free hashing *if there exists a probabilistic polynomial-time algorithm I so that the following holds*

1. (admissible indexing – technical):[4] *For some polynomial p, all sufficiently large n's and every s in the range of $I(1^n)$ it holds that $n \leq p(|s|)$.*

---

[3] This observation only saves us a polynomial factor in the forging probability. That is, if A did not know which part of the forged complex-signature to use in its own forgery, it could have selected one at random (and be correct with $1/\text{poly}(n)$ probability).

[4] This condition is made merely to avoid annoying technicalities. Note that $|s| = \text{poly}(n)$ holds by definition of I.

2. (efficient evaluation): *There exists a polynomial-time algorithm that given s and x, returns $h_s(x)$.*

3. (hard to form collisions): *We say that the pair (x, x') forms a* collision under the function h *if $h(x) = h(x')$ but $x \neq x'$. We require that every probabilistic polynomial-time algorithm, given $I(1^n)$ as input, outputs a collision under $h_{I(1^n)}$ with negligible probability. That is, for every probabilistic polynomial-time algorithm A, every polynomial p and all sufficiently large n's,*

$$\Pr\left[A(I(1^n)) \text{ is a collision under } h_{I(1^n)}\right] < \frac{1}{p(n)}$$

where the probability is taken over the internal coin tosses of algorithms I and A.

The function ℓ is called the range specifier *of the collection.*

Note that the range specifier must be super-logarithmic (or else one may easily find a collisions by selecting $2^{\ell(n)} + 1$ preimages and computing there image under the function). In Section 6.2.3, we show how to construct collision-free hashing functions using claw-free collections. But first, we show how to use the former in order to convert a length-restricted signature scheme into a full-fledged one.

Construction 6.2.6 (hash and sign): *Let ℓ and (G, S, V) be as in Theorem 6.2.2, and let $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}_{r \in \{0,1\}^*}$ be as in Definition 6.2.5. We construct a general signature scheme, (G', S', V'), as follows:*

key-generation with G': *On input 1^n, algorithm G' first invokes G to obtain $(s, v) \leftarrow G(1^n)$. Next it invokes I, the indexing algorithm of the collision-free hashing collection, to obtain $r \leftarrow I(1^n)$. Finally, G' outputs the pair $((r, s), (r, v))$, where (r, s) serves as a signing-key and (r, v) serves as a verification-key.*

signing with S': *On input a signing-key $(r, s) \in G'_1(1^n)$ and a document $\alpha \in \{0,1\}^*$, algorithm S' invokes S once to produce and output $S_s(h_r(\alpha))$.*

verification with V': *On input a verifying-key $(r, v) \in G'_2(1^n)$, a document $\alpha \in \{0,1\}^*$, and a alledged signature β, algorithm V' invokes V, and outputs $V_v(h_r(\alpha), \beta)$.*

Proposition 6.2.7 *Suppose that (G, S, V) is an ℓ-restricted signature scheme that is secure in the private-key* (resp., public-key) *model. Suppose that $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}_{r \in \{0,1\}^*}$ is indeed a collision-free hashing collection. Then (G', S', V'), as defined in Construction 6.2.6 is a full-fledged signature scheme that is secure in the private-key* (resp., public-key) *model.*

Proof: Intuitively, the security of (G', S', V') follows from the security of (G, S, V) and the collision-freeness property of the collection $\{h_r\}$. Specifically, forgery relative to (G', S', V') can be obtained by either a forged S-signature to a hash-value different from all hash-values that appeared in the attack or by forming a collision under the hash function. That is, the actual proof is by a reducibility argument. Given an adversary A' attacking the complex scheme (G', S', V'), we construct an adversary A that attacks the ℓ-restricted scheme, (G, S, V), as well as an algorithm B forming collisions under the hashing collection $\{h_r\}$. Both A and B will have running-time related to that of A'. We show if A' is successful with non-negligible probability than the same holds for either A or B. Thus, in either case, we recah a contradiction. We start with the description of algorithm A, which is designed to attack the ℓ-restricted scheme (G, S, V).

The adversary A operates as follows. First it uses I (the indexing algorithm of the collision-free hashing collection) to obtain $r \leftarrow I(1^n)$, exactly as done in the second step of G'. Next, it uses r and its oracle S_s in order to emulate the oracle $S'_{r,s}$ for A'. This can be done in a straightforward manner; that is, algorithm A will act as $S'_{r,s}$ does using the oracle S_s. When A' outputs a document-signature pair relative to the complex scheme (G', S', V'), algorithm A tries to use this pair in order to form a document-signature pair relative to the ℓ-restricted scheme, (G, S, V).

We stress again that from the point of view of adversary A', the distribution of keys and oracle answers that A provides it with is exactly as in a real attack of A' on (G', S', V'). This is a crucial point since we use the fact that events that occur in a real attack of A' on (G', S', V'), occur with the same probability in the emulation of (G', S', V') by A.

Assume that with (non-negligible) probability $\varepsilon'(n)$, the (probabilistic polynomial-time) algorithm A' succeeds in existentially forging relative to the complex scheme (G', S', V'). We consider the following cases regarding the forging event, letting (α_i, β_i) denote the i^{th} query and answer pair made by A', and (α, β) denote the forged document-signature pair that A' outputs (in case of success):

1. $h_r(\alpha) \neq h_r(\alpha_i)$ for all i's. (That is, the hash value used in the forged signature is different from all hash values used in the queries to S_s.) In this case, the pair $(h_r(\alpha), \beta)$ constitutes a success in existential forgery relative to the ℓ-restricted scheme.

2. $h_r(\alpha) = h_r(\alpha_i)$ for some i. (That is, the hash value used in the forged signature equals the hash value used in the i^{th} query to S_s, although $\alpha \neq \alpha_i$.) In this case, the pair (α, α_i) forms a collision under h_r (and we do not obtain success in existential forgery relative to the ℓ-restricted scheme).

Thus, if the first case occurs with probability at least $\varepsilon'(n)/2$ then A succeeds in its attack on (G, S, V) with probability at least $\varepsilon'(n)/2$, which contradicts the security of the ℓ-restricted scheme (G, S, V). On the other hand, if the second case occurs with probability at least $\varepsilon'(n)/2$ then we derive a contradiction to the collision-freeness of the hashing collection $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}_{r\in\{0,1\}^*}$. Details (regarding the second case) follow.

We construct an algorithm, B, that given $r \leftarrow I(1^n)$, attempts to form collisions under h_r as follows. On input r, algorithm B generates $(s, v) \leftarrow G(1^n)$, and emulates the attack of A on this instance of the ℓ-restricted scheme, with the exception that B does not invoke algorithm I to obtain an index of a hash function but rather uses the index r. Finally, when A' (emulated by A) outputs a forged signature, algorithm B checks if the second case occurs in which case it obtains and outputs a collision under h_r.

We stress that from the point of view of the emulated adversary A, the execution is distributed exactly as in its attack on (G, S, V). Thus, since the second case above occurs with probability at least $\varepsilon'(n)/2$ in a real attack, it follows that B succeeds to form a collision under $h_{I(1^n)}$ with probability at least $\varepsilon'(n)/2$. This contradicts the collision-freeness of the hashing functions, and the proposition follows. ■

Using the hashing paradigm in practice: The *hash-then-sign paradigm*, underlying Construction 6.2.6, is often used in practice. Specifically, a document is signed using a two-stage process: first the document is hashed into a (relatively) short bit string, and next a basic signature scheme is applied to the resulting string. We stress that this process yields a secure signature scheme only if the hashing scheme is *collision-free* (as defined above). In Section 6.2.3, we present a way to construct collision-free hashing functions. Alternatively, one may indeed postulate that certain off-the-shelf products (such as MD5 or SHA) are collision-free, but such assumptions need to be seriously examined (and indeed may turn out false). We stress that using a hashing scheme, in the above two-stage process, without seriously evaluating whether it is collision-free is a very dangerous practice.

6.2.3 * Constructing collision-free hashing functions

In this subsection we show how to construct collision-free hashing functions using a claw-free collection of permutations as defined in Section 2.4.5. Recall that such a collection consists of pairs of permutations, (f_s^0, f_s^1), so that both f_s^σ's are permutations over a set D_s and of a probabilistic polynomial-time index selection algorithm I so that

1. *The domain is easy to sample*: there exists a probabilistic polynomial-time algorithm that given s outputs a string uniformly distributed over D_s.

2. *The permutations are easy to evaluate*: there exists a polynomial-time algorithm that given s, σ and $x \in D_s$, outputs $f_s^\sigma(x)$.

3. *Hard to form claws*: every probabilistic polynomial-time algorithm, given $s \leftarrow I(1^n)$ outputs a pair (x, y) so that $f_s^0(x) = f_s^1(y)$ with at most negligible probability. That is, a pair (x, y) satisfying $f_s^0(x) = f_s^1(y)$ is called

a claw for index s, and C_s denote the set of claws for index s. Then, it is required that for every probabilistic polynomial-time algorithm, A', every positive polynomial $p(\cdot)$, and all sufficiently large n's

$$\Pr\left[A'(I(1^n)) \in C_{I(1^n)}\right] < \frac{1}{p(n)}$$

Note that since f_s^0 and f_s^1 are permutations over the same set, many claws do exists (i.e., $|C_s| = |D_s|$). However, the third item above postulates that for s generated by $I(1^n)$ such claws are hard to find. We may assume, without loss of generality, that for some $\ell : \mathbb{N} \to \mathbb{N}$ and all s's it holds that $D_s \subseteq \{0,1\}^{\ell(|s|)}$. Indeed, ℓ must be polynomially bounded. For simplicity we assume that $I(1^n) \in \{0,1\}^n$. Recall that such collections of permutation pairs can be constructed based on the standard DLP or factoring intractability assumptions (see Section 2.4.5).

Construction 6.2.8 (collision-free hashing based on claw-free permutations pairs): *Given an index selecting algorithm I for a collection of permutation pairs $\{(f_s^0, f_s^1)\}_s$ as above, we construct a collection of hashing functions $\{h_{(s,r)} : \{0,1\}^* \to \{0,1\}^{|r|}\}_{(s,r)\in\{0,1\}^*\times\{0,1\}^*}$ as follows:*

index selection algorithm: *On input 1^n, we first invoke I to obtain $s \leftarrow I(1^n)$, and next use the domain sampler to obtain a string r that is uniformly distributed in D_s. We output the index (s,r), defining a hashing function*

$$h_{(s,r)}(x) \stackrel{\text{def}}{=} f_s^{y_1} f_s^{y_2} \cdots f_s^{y_t}(r)$$

where $y_1 \cdots y_t$ is a prefix-free encoding of x; that is, for any $x \neq x'$ the coding of x is not a prefix of the coding of x'. For example, code $x_1x_2\cdots x_m$ by $x_1x_1x_2x_2\cdots x_mx_m01$.

evaluation algorithm: *Given an index (s,r) and a string x, we compute $h_{(s,r)}(x)$ in a straightforward manner. That is, first we compute the prefix-free encoding of x, denoted $y_1 \cdots y_t$. Next, we use the evaluation algorithm of the claw-free collection to compute $f_s^{y_1} f_s^{y_2} \cdots f_s^{y_t}(r)$, which is the desired output.*

Proposition 6.2.9 *Suppose that the collection of permutation pairs $\{(f_s^0, f_s^1)\}_s$ together with the index selecting algorithm I constitute a claw-free collection. Then, the function ensemble $\{h_{(s,r)} : \{0,1\}^* \to \{0,1\}^{|r|}\}_{(s,r)\in\{0,1\}^*\times\{0,1\}^*}$ as defined in Construction 6.2.8 constitute a collision-free hashing with a range specifying function ℓ' satisfying $\ell'(n + \ell(n)) = \ell(n)$.*

Proof: The proof is by a reducibility argument. Given an algorithm A' that, on input (s,r), forms a collision under $h_{(s,r)}$, we construct an algorithm A that on input s forms a claw for index s.

On input s (supposedly generated by $I(1^n)$), algorithm A selects r uniformly in D_s, and invokes algorithm A' on input (s,r). Suppose that A' outputs a pair (x,x') so that $h_{(s,r)}(x) = h_{(s,r)}(x')$ but $x \neq x'$. Without loss of generality,[5] assume that the coding of x equals $y_1 \cdots y_{i-1} 0 z_{i+1} \cdots z_t$, and that the coding of x' equals $y_1 \cdots y_{i-1} 1 z'_{i+1} \cdots z'_{t'}$. By the definition of $h_{(s,r)}$, it follows that

$$f_s^{y_1} \cdots f_s^{y_{i-1}} f_s^0 f_s^{z_{i+1}} \cdots f_s^{z_t}(r) = f_s^{y_1} \cdots f_s^{y_{i-1}} f_s^1 f_s^{z'_{i+1}} \cdots f_s^{z'_{t'}}(r) \tag{6.1}$$

Since each of the f_s^σ's is 1-1, Eq. (6.1) implies that

$$f_s^0 f_s^{z_{i+1}} \cdots f_s^{z_t}(r) = f_s^1 f_s^{z'_{i+1}} \cdots f_s^{z'_{t'}}(r) \tag{6.2}$$

Computing $w \stackrel{\rm def}{=} f_s^{z_{i+1}} \cdots f_s^{z_t}(r)$ and $w' \stackrel{\rm def}{=} f_s^{z'_{i+1}} \cdots f_s^{z'_{t'}}(r)$, algorithm A obtains a pair (w,w') so that $f_s^0(w) = f_s^1(w')$. Thus, algorithm A forms claws for index $I(1^n)$ with probability that is bounded below by the probability that A' forms a collision under $h_{I'(1^n)}$, where I' is the index selection algorithm as defined in Construction 6.2.8. Using the hypothesis that the collection of pairs (together with I) is claw-free, the proposition follows. ■

6.3 Constructions of Message Authentication Schemes

In this section we present several constructions of secure message authentication schemes (referred to above as secure private-key signature schemes). Below, we sometimes refer to a message authentication scheme as to a Message Authentication Code (which is the traditional term), abbreviated by MAC.

6.3.1 By using pseudorandom functions

Message authentication schemes can be easily constructed using pseudorandom functions (as defined in Section 3.6). Specifically, by Theorem 6.2.2, it suffices to construct an ℓ-restricted message authentication scheme, for any superlogarithmically growing ℓ.

Construction 6.3.1 (an ℓ-restricted MAC based on pseudorandom functions): *Let ℓ be a superlogarithmically growing function, and $\{f_s : \{0,1\}^{\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ be as in Definition 3.6.4. We construct an ℓ-restricted message authentication scheme, (G,S,V), as follows:*

key-generation with G: *On input 1^n, we uniformly select $s \in \{0,1\}^n$, and output the key-pair (s,s).*

(Indeed, the verification-key equals the signing-key.)

[5] Let $C(x)$ (resp., $C(x')$) denote the prefix-free coding of x (resp., x'). Then $C(x)$ is not a prefix of $C(x')$, and $C(x')$ is not a prefix of $C(x)$. It follows that $C(x) = uv$ and $C(x') = uv'$, where v and v' differ in their leftmost bit. Without loss of generality, we may assume that the leftmost bit of v is is 0, and the leftmost bit of v' is 1.

signing with S: *On input a signing-key $s \in \{0,1\}^n$ and an $\ell(n)$-bit string α, we compute and output $f_s(\alpha)$ as a signature of α.*

verification with V: *On input a verification-key $s \in \{0,1\}^n$, an $\ell(n)$-bit string α, and an alledge signature β, we accept if and only if $\beta = f_s(\alpha)$.*

Analogous constructions can be presented using the generalized notions of pseudorandom functions defined in Definitions 3.6.9 and 3.6.12. In particular, using a pseudorandom function ensemble of the form $\{f_s : \{0,1\}^* \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, we obtain a general message authentication scheme (rather than a length-restricted one). We prove only the security of the ℓ-restricted message authentication scheme of Construction 6.3.1. (The security of the general message authentication scheme can be established analogously; see Exercise 2.)

Proposition 6.3.2 *Suppose that $\{f_s : \{0,1\}^{\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is a pseudorandom function, and that ℓ is a superlogarithmically growing function, Then Construction 6.3.1 constitutes a secure ℓ-restricted message authentication scheme.*

Proof: The proof follows the general methodology suggested in Section 3.6.3. Specifically, we consider the security of an ideal scheme in which the pseudorandom function is replaced by a truely random function (mapping $\ell(n)$-bit long strings to $\ell(n)$-bit long strings). Clearly, an adversary that obtains the values of this *random* function at arguments of its choice, cannot predict its value at a new point with probability greater than $2^{-\ell(n)}$. Thus, an adversary attacking the *ideal scheme* may succeed in existential forgery with at most negligible probability. The same must hold for any efficient adversary that attacks the *actual scheme*, since otherwise such an adversary yields a violation of the pseudorandomness of $\{f_s : \{0,1\}^{\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$. Details follow.

The actual proof is by a reducibility argument. Given a probabilistic polynomial-time A attacking the scheme (G, S, V), we consider what happens when A is attacking an ideal scheme in which a random function is used instead of a pseudorandom one. That is, we refer to two experiments:

1. *A attacks the actual scheme*: On input 1^n, machine A is given oracle access to $f_s : \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$, where s is uniformly selected in $\{0,1\}^n$. After making some queries of its choice, A outputs a pair (α, β), where α is different from all its queries, and is deem successful if and only if $\beta = f_s(\alpha)$.

2. *A attacks the ideal scheme*: On input 1^n, machine A is given oracle access to a function $F : \{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$, uniformly selected among all such possible functions. After making some queries of its choice, A outputs a pair (α, β), where α is different from all its queries, and is deem successful if and only if $\beta = F(\alpha)$.

   Clearly, A's success probability in this experiment is at most $2^{-\ell(n)}$, which is a negligible function (since ℓ is super-logarithmic).

Assuming that A's success probability in the actual attack is non-negligible, we derive a contradiction to the pseudorandomness of the function ensemble $\{f_s\}$. Specifically, we consider a distinguisher D that on input 1^n and oracle access to a function $f:\{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$, behaves as follows: First D emulates the actions of A, while answering A's queries using its oracle f. When A outputs a pair (α,β), the distinguisher makes one additional oracle query to f and outputs 1 if and only if $f(\alpha)=\beta$.

Note that when f is selected uniformly among all possible $\{0,1\}^{\ell(n)} \to \{0,1\}^{\ell(n)}$ functions, D emulates an attack of A on the ideal scheme, and thus outputs 1 with negligible probability (as explained above). On the other hand, if f is uniformly selected in $\{f_s\}_{s\in\{0,1\}^n}$ then D emulates an attack of A on the actual scheme, and thus (due to the contradiction hypothesis) outputs 1 with non-negligible probability. We reach a contradiction to the pseudorandomness of $\{f_s\}_{s\in\{0,1\}^n}$. The proposition follows. ■

A plausibility result: Combining Theorem 6.2.2, Proposition 6.3.2, and Corollary 3.6.7, it follows that the existence of one-way functions implies the existence of message authentication schemes. The converse also holds; see Exercise 3. Thus, we have:

Theorem 6.3.3 *Secure message authentication schemes exist if and only if one-way functions exist.*

6.3.2 * Other alternatives

Author's Note: *a second look at hash-then-mac vs direct-PRF*

Author's Note: *.... and more efficient schemes may be obtained based on other cryptographic primitives. Work out the following paragraph....*

Fingerprinting the message using a scheme which is *secure against forgery provided that the adversary does not have access to the scheme's outcome* (e.g., using Universal Hashing [48]), and "*hiding*" the result using a *non-malleable* scheme (e.g., a private-key encryption or a pseudorandom function). (Non-malleability is not required in certain cases; see [208].)

6.4 Constructions of Signature Schemes

In this section we present several constructions of secure public-key signature schemes. Here we refer to such schemes as *signature schemes*, which is indeed the traditional term.

Two central paradigms in the construction of *signature schemes* are the "refreshing" of the "effective" signing-key, and the usage of an "authentication tree". In addition, the "hashing paradigm" (employed also in the construction of message authentication schemes), plays a even more central role in the following presentation. In addition to the above, we use the notion of *one-time signature scheme* defined in Section 6.4.1.

The current section is organized as follows. In Section 6.4.1 we define and construct various types of one-time signature schemes. The "hashing paradigm" plays a central role in one of these constructions, which in turn is essential for Section 6.4.2. In Section 6.4.2 we show how to use one-time signature schemes to construct general signature schemes. This construction utilizes "refreshing paradigm" (as employed to one-time signature schemes) and an "authentication tree". In Section 6.4.3, we define Universal One-Way Hashing and show how to use it (in the previous constructions) instead of collision-free hashing. The gain in using Universal One-Way Hashing (rather than collision-free hashing) is that the former can be constructed based on any one-way function (whereas this is not known for collision-free hashing). Thus, we obtain:

Theorem 6.4.1 *Secure signature schemes exist if and only if one-way functions exist.*

The difficult direction is to show that the existence of one-way functions implies the existence of signature schemes. For the other direction, see Exercise 3.

6.4.1 One-time signature schemes

In this section we define and construct various types of one-time signature schemes. Specifically, we first define one-time signature schemes, next a length-restricted version of this notion (analogous to Definition 6.2.1), then present a simple construction of the latter, and finally show how such a construction combined with collision-free hashing yields a general one-time signature scheme.

6.4.1.1 Definitions

Loosely speaking, one-time signature schemes are signature schemes for which the security requirement is restricted to attacks in which the adversary asks for at most one string to be signed. That is, the mechanics of one-time signature schemes are as of ordinary signature schemes (see Definition 6.1.1), but the security requirement is relaxed as follows.

- A chosen one-message attack is a process that can obtain a signature to *at most one* string of its choice. That is, the attacker is given v as input, and obtains a signature relative to s, where $(s,v) \leftarrow G(1^n)$ for an adequate n.

  (Note that in this section we focus on public-key signature schemes and thus we present only the definition for this case.)

- Such an attack is said to succeeds (*in existential forgery*) if it outputs a valid signature to a string for which it has NOT requested a signature during the attack.

  (Indeed, the notion of success is exactly as in Definition 6.1.2.)

- A one-time signature scheme is secure (or unforgeable) if every probabilistic polynomial-time chosen *one*-message attack succeeds with at most negligible probability.

Moving to the formal definition, we again model a chosen message attack as a probabilistic oracle machine; however, since here we are only about *one*-message attacks, we consider only oracle machines that make at most one query. Let M be such a machine. As before, we denote by $Q_M^O(x)$ the set of queries made by M on input x and access to oracle O, and let $M_1^O(x)$ denote the first string in the output of M on input x and access to oracle O. Note that here $|Q_M^O(x)| \leq 1$ (i.e., M may either make no queries or a single query).

Definition 6.4.2 (security for one-time signature schemes): *A one-time signature scheme is* secure *if for every probabilistic polynomial-time oracle machine* M that makes at most one query, *every polynomial p and all sufficiently large n, it holds that*

$$\mathsf{Pr}\left[\begin{array}{c} V_{G_2(1^n)}(M^{S_{G_1(1^n)}}(G_2(1^n)))=1 \\ and \\ M_1^{S_{G_1(1^n)}}(G_2(1^n)) \notin Q_M^{S_{G_1(1^n)}}(G_2(1^n)) \end{array}\right] < \frac{1}{p(n)}$$

where the probability is taken over the coin tosses of algorithms G, S and V as well as over the coin tosses of machine M.

We now define a length-restricted version of one-time signature schemes. The definition is indeed analogous to Definition 6.2.1:

Definition 6.4.3 (length-restricted one-time signature schemes): *Let $\ell : \mathbb{N} \to \mathbb{N}$. An ℓ-*restricted one-time signature scheme *is a triple, (G, S, V), of probabilistic polynomial-time algorithms satisfying the the mechanics of Definition 6.2.1. That is, it satisfies the following two conditions*

1. *As in Definition 6.1.1, on input 1^n, algorithm G outputs a pair of bit strings.*
2. *Analogously to Definition 6.1.1, for every n and every pair (s, v) in the range of $G(1^n)$, and for every $\alpha \in \{0,1\}^{\ell(n)}$, algorithms S and D satisfy* $\mathsf{Pr}[V(v, \alpha, S(s, \alpha))=1] = 1$.

Such a scheme is called secure (in the one-time model) *if the requirement of Definition 6.4.2 holds when restricted to attackers that only make queries of length $\ell(n)$ and output a pair (α, β) with $|\alpha| = \ell(n)$. That is, we consider only attackers that make at most one query, this query has to be of length $\ell(n)$, and the output (α, β) must satisfy $|\alpha| = \ell(n)$.*

6.4.1.2 Length-restricted one-time signature schemes

We now present a simple construction of length-restricted one-time signature schemes. The construction works for any length restriction function ℓ, but the keys will have length greater than ℓ. The latter fact limits the applicability of such schemes, and will be removed in the next subsection. But first, we construct ℓ-restricted one-time signature schemes based on any one-way function f. We may assume for simplicity that f is length preserving.

Construction 6.4.4 (an ℓ-restricted one-time signature scheme): *Let $\ell : \mathbb{N} \to \mathbb{N}$ be polynomially-bounded and polynomial-time computable, and $f : \{0,1\}^* \to \{0,1\}^*$ be polynomial-time computable and length-preserving. We construct an ℓ-restricted one-time signature scheme, (G, S, V), as follows:*

key-generation with G: *On input 1^n, we uniformly select $s_1^0, s_1^1, ..., s_{\ell(n)}^0, s_{\ell(n)}^1 \in \{0,1\}^n$, and compute $v_i^j = f(s_i^j)$, for $i = 1, ..., \ell(n)$ and $j = 0, 1$. We let $s = ((s_1^0, s_1^1), ..., (s_{\ell(n)}^0, s_{\ell(n)}^1))$, and $v = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$, and output the key-pair (s, v).*

(Note that $|s| = |v| = 2 \cdot \ell(n) \cdot n$.)

signing with S: *On input a signing-key $s = ((s_1^0, s_1^1), ..., (s_{\ell(n)}^0, s_{\ell(n)}^1))$ and an $\ell(n)$-bit string $\alpha = \sigma_1 \cdots \sigma_{\ell(n)}$, we output $(s_1^{\sigma_1}, ..., s_{\ell(n)}^{\sigma_{\ell(n)}})$ as a signature of α.*

verification with V: *On input a verification-key $v = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$, an $\ell(n)$-bit string $\alpha = \sigma_1 \cdots \sigma_{\ell(n)}$, and an alledge signature $\beta = (\beta_1, ..., \beta_{\ell(n)})$, we accept if and only if $v_i^{\sigma_i} = f(\beta_i)$, for $i = 1, ..., \ell(n)$.*

Proposition 6.4.5 *If f is a one-way function then Construction 6.4.4 constitutes a secure ℓ-restricted one-time signature scheme.*

Note that Construction 6.4.4 does NOT constitute a (general) ℓ-restricted signature scheme: An attacker that obtains signatures to *two* strings (e.g., to the strings $0^{\ell(n)}$ and $1^{\ell(n)}$), can present a valid signature to any $\ell(n)$-bit long string (and thus totally break the system). However, here we consider only attackers that may ask for at most one string (of their choice) to be signed. As a corollary to Proposition 6.4.5, we obtain:

Corollary 6.4.6 *If there exist one-way functions then, for every polynomially-bounded and polynomial-time computable $\ell : \mathbb{N} \to \mathbb{N}$, there exist secure ℓ-restricted one-time signature schemes.*

Proof of Proposition 6.4.5: Intuitively, forging a signature (after seeing at most one signature to a different message) requires inverting f on some image. The actual proof is by a reducibility argument. Given an adversary A attacking the scheme (G, S, V), while making at most one query, we construct an algorithm A' for inverting f.

As a warm-up, let us first deal with the case in which A makes no queries at all. In this case, on input y (supposedly in the range of f), algorithm A' proceeds as follows. First A' selects p uniformly in $\{1, ..., \ell(n)\}$, q uniformly in $\{0, 1\}$, and

$s_1^0, s_1^1, ..., s_{\ell(n)}^0, s_{\ell(n)}^1$ each independently and uniformly in $\{0,1\}^n$. (Actaully, s_p^q is not used and needs not be selected.) For every $i \in \{1, ..., \ell(n)\} \setminus \{p\}$, and every $j \in \{0,1\}$, algorithm A' computes $v_i^j = f(s_i^j)$. Algorithm A' also computes $v_p^{1-q} = f(s_p^{1-q})$, and sets $v_p^q = y$ and $v = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$. Note that if $y = f(x)$, for a uniformly distributed $x \in \{0,1\}^n$, then for each possible choice of p and q, the sequence v is distributed identically to the public-key generated by $G(1^n)$. Next, A' invokes A on input v, hoping that A will forge a signature, denoted $\beta = \tau_1 \cdots \tau_{\ell(n)}$, to a message $\alpha = \sigma_1 \cdots \sigma_{\ell(n)}$ so that $\sigma_p = q$. If this event occurs, A' obtains a preimage of y under f, since the validity of the signature implies that $f(\tau_p) = v_p^{\sigma_p} = v_p^q = y$. Observe that conditioned on the value of v and the internal coin tosses of A, the value q is uniformly distributed in $\{0,1\}$. Thus, A' inverts f with probability $\varepsilon(n)/2$, where $\varepsilon(n)$ denotes the probability that A succeeds in forgery.

We turn back to the actual case in which A may make a single query. (Without loss of generality, we may assume that A always makes a single query; see Exercise 4.) In this case, on input y (supposedly in the range of f), algorithm A' selects p, q and the s_i^j's, and forms the v_i^j's and v exactly as in the warm-up above.[6] Recall that if $y = f(x)$, for a uniformly distributed $x \in \{0,1\}^n$, then for each possible choice of p and q, the sequence v is distributed identically to the public-key generated by $G(1^n)$. Also note that for each v_i^j other than $v_p^q = y$, algorithm A' holds a random preimage under f. Next, A' invokes A on input v, and tries to answer its query, denoted $\alpha = \sigma_1 \cdots \sigma_{\ell(n)}$. We consider two cases regarding the signature required by A.

1. If $\sigma_p = q$ then A' can not supply the desired signature since it lacks a preimage of $y = s_p^q$ under f. Thus, in this case A' aborts. However, this case occurs with probability $\frac{1}{2}$, independently of the actions of A (since v yields no information on either p or q).

   (That is, conditioned on the value of v and the internal coin tosses of A, this case occurs with probability $\frac{1}{2}$.)[7]

2. If $\sigma_p = 1 - q$ then A' can supply the desired signature since it holds all the relevant s_i^j's (i.e., random preimages of the relevant v_i^j's under f). In particular, A' holds both s_i^j's, for $i \neq p$, as well as s_p^{1-q}. Thus, A' answers with $(s_1^{\sigma_1}, ..., s_{\ell(n)}^{\sigma_{\ell(n)}})$.

Note that conditioned on the value of v, the internal coin tosses of A and on the second case occuring, p is uniformly distributed in $\{1, ..., \ell(n)\}$. When the second

[6] That is, first A' selects p uniformly in $\{1, ..., \ell(n)\}$, q uniformly in $\{0,1\}$, and $s_1^0, s_1^1, ..., s_{\ell(n)}^0, s_{\ell(n)}^1$ each independently and uniformly in $\{0,1\}^n$. For every $i \in \{1, ..., \ell(n)\} \setminus \{p\}$, and every $j \in \{0,1\}$, algorithm A' computes $v_i^j = f(s_i^j)$. Algorithm A' also computes $v_p^{1-q} = f(s_p^{1-q})$, and sets $v_p^q = y$ and $v = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$.

[7] This follows from an even stronger statement by which conditioned on the value of v, the internal coin tosses of A and on the value of p, the current case happens with probability $\frac{1}{2}$. The stronger statement holds since conditioned on all the above, q is uniformly distributed in $\{0,1\}$ (and so $\sigma_p = q$ happens with probability exactly $\frac{1}{2}$).

case occurs, A obtains a signature to α and this signature is distributed exactly as in a real attack. We stress that since A asks at most one query, no additional query will be asked by A. Also note that, in this case (i.e., $\sigma_p = 1-q$), algorithm A outputs a forged mesasage–signature pair, denoted (α', β'), with probability exactly as in a real attack.

For simplicity we assume below that A has indeed made a single query α (otherwise one may consider α and the σ_i's to be some non-boolean dummy values and apply the following reasoning nevertheless).[8] Let $\alpha' = \sigma'_1 \cdots \sigma'_{\ell(n)}$ and $\beta' = s'_1 \cdots s'_{\ell(n)}$, where (α', β') is the forged mesasage–signature pair output by A. By our hypothesis (that this is a forgery-success event) it follows that $\alpha' \neq \alpha$ and that $f(s'_i) = v_i^{\sigma'_i}$ for all i's. Since (conditioned on all the above) p is uniformly distributed in $\{1, ..., \ell(n)\}$, it follows that with probability $\frac{|\{i:\sigma'_i \neq \sigma_i\}|}{\ell(n)} \geq \frac{1}{\ell(n)}$ it holds that $\sigma'_p \neq \sigma_p$, and then A' obtains a preimage of y under f (since s'_p satisfies $f(s'_p) = v_p^{\sigma'_p}$, which in turn equals $v_p^{1-\sigma_p} = v_p^q = y$).

To summarize, assuming that A succeeds in a single-message attack on (G, S, V) with probability $\varepsilon(n)$, algorithm A' inverts f on a random image (i.e., on $f(U_n)$) with probability

$$\varepsilon(n) \cdot \frac{1}{2} \cdot \frac{|\{i : \sigma'_i \neq \sigma_i\}|}{\ell(n)} \geq \frac{\varepsilon(n)}{2\ell(n)}$$

Thus, if A is a probabilistic polynomial-time *chosen one-message* attack that forges signatures with non-negligible probability then A' is a probabilistic polynomial-time algorithm that inverts f with non-negligible probability (in violation of the hypothesis that f is a one-way function). The proposition follows. ■

6.4.1.3 From length-restricted schemes to general ones

We now combine a length-restricted one-time signature scheme with collision-free hashing to obtain a general one-time signature scheme. The construction is identical to Construction 6.2.6, except that here (G, S, V) is an ℓ-restricted *one-time* signature scheme rather than an ℓ-restricted (general) signature scheme. Analogously to Proposition 6.2.7, we obtain.

Proposition 6.4.7 *Suppose that (G, S, V) is a secure ℓ-restricted one-time signature scheme, and that $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}_{r \in \{0,1\}^*}$ is a collision-free hashing collection. Then (G', S', V'), as defined in Construction 6.2.6 is a secure one-time signature scheme.*

Proof: The proof is identical to the proof of Proposition 6.2.7; we merely notice that if the adversary A', attacking (G', S', V'), makes at most one query then the same holds for the adversary A that we constract to attack (G, S, V). In

[8] Alternatively, recall that, without loss of generality, we may assume that A always makes a single query; see Exercise 4.

general, the adversary A constructed in the proof of Proposition 6.2.7 makes a one query per each query of the adversary A'. ■

Combining Proposition 6.4.7, Corollary 6.4.6, and the fact that collision-free hashing collections imply one-way functions (see Exercise 7), we obtain:

Corollary 6.4.8 *If there exist collision-free hashing collections then there exist secure one-time signature schemes.*

Comments: We stress that when using Construction 6.2.6, signing each document under the (general) scheme (G', S', V') requires signing a single string under the ℓ-restricted scheme (G, S, V). This is in contrast to Construction 6.2.3 in which signing a document under the (general) scheme (G', S', V') requires signing many strings under the ℓ-restricted scheme (G, S, V), where the number of such strings depends (linearly) on the length of the original document.

Construction 6.2.6 calls for the use of collision-free hashing. The latter can be constructed using any claw-free permutation collection (see Proposition 6.2.9), however it is not know whether collision-free hashing can be constructed based on any one-way function. Wishing to construct signature schemes based on any one-way function, in Section 6.4.3 we avoid the use of collision-free hashing. Instead, we use "universal one-way hashing functions" (to be defined), and present a varaint of Construction 6.2.6 that uses these functions rather than collision-free ones.

6.4.2 From one-time signature schemes to general ones

In this section we show how to construct general signature schemes using one-time signature schemes. That is, we shall prove:

Theorem 6.4.9 *If there exist secure one-time signature schemes then secure signature schemes exist as well.*

Actually, we can use length-restricted one-time signature schemes, *provided that the length of the strings being signed is at least twice the length of the verification key.* Unfortunately, Construction 6.4.4 does not satisfy this condition. Nevertheless, Corollary 6.4.8 does provide one-time signature schemes. Thus, combining Theorem 6.4.9 and Corollary 6.4.8, we obtain:

Corollary 6.4.10 *If there exist collision-free hashing collections then there exist secure signature schemes.*

Note that Corollary 6.4.10 asserts the existence of secure (public-key) signature schemes, based on an assumption that does *not* mention trapdoors. We stress this point because of the contrast to the situation with respect to public-key encryption schemes, where a trapdoor property seem necessary for the construction of secure schemes.

6.4.2.1 The refreshing paradigm

The so-called "refreshing paradigm" plays a central role in the proof of Theorem 6.4.9. Loosely speaking, the "refreshing paradigm" suggests to reduce the dangers of a chosen message attack on the signature scheme by using "fresh" instances of the scheme for signing. Of course, these fresh instances should be authenticated by the original instance (corresponding to the verification-key that is publically known).

Example: To demonstrate the refreshing paradigm, consider a basic signature scheme (G, S, V) used as follows. Suppose that the user U has generated a key-pair, $(s, v) \leftarrow G(1^n)$, and has placed the verification-key v on a public-file. When a party asks U to sign some document α, the user U generates a new (fresh) key-pair, $(s', v') \leftarrow G(1^n)$, signs v' using the original signing-key s, signs α using the new (fresh) signing-key s', and presents $(S_s(v'), v', S_{s'}(\alpha))$ as a signature to α. An alledged signature, (β_1, v', β_2), is verified by checking whether both $V_v(v', \beta_1) = 1$ and $V_{v'}(\alpha, \beta_2) = 1$. Intuitively, the gain in terms of security is that a full-fledged chosen message attack cannot be lounched on (G, S, V). All that an attacker may obtain (via a chosen message attack on the new scheme) is signatures, relative to the original signing-key s, to randomly chosen strings (taken from the distribution $G_2(1^n)$) as well as additional signatures each relative to a random and independently chosen signing-key.

We refrain from analyzing the features of the signature scheme presented in the above example. Instead, as a warm-up to the actual construction used in the next section (in order to establish Theorem 6.4.9), we present and analyze a similar construction (which is – in some sense – a hybrid of the two constructions). The reader may skip this warm-up, and proceed directly to Section 6.4.2.2.

Construction 6.4.11 (a warm-up): *Let (G, S, V) be a signature scheme and (G', S', V') be a one-time signature scheme. Consider a signature scheme, (G'', S'', V''), with $G'' = G$, as follows:*

signing with S'': *On input a signing-key s and a document $\alpha \in \{0,1\}^*$, first invoke G' to obatin $(s', v') \leftarrow G'(1^n)$. Next, invoke S to obtain $\beta_1 \leftarrow S_s(v')$, and S' to obatin $\beta_2 \leftarrow S'_{s'}(\alpha)$. The final output is (β_1, v', β_2).*

verification with V'': *On input a verifying-key v, a document $\alpha \in \{0,1\}^*$, and a alledged signature $\beta = (\beta_1, v', \beta_2)$, we output 1 if and only if both $V_v(v', \beta_1) = 1$ and $V'_{v'}(\alpha, \beta_2) = 1$.*

Construction 6.4.11 differs from the above example only in that a one-time signature scheme is used to generate the "second signature" (rather than using the same ordinary signature scheme). The use of a one-time signature scheme is natural here, since it is unlikely that the same signing-key s' will be selected in two invocations of S''.

Proposition 6.4.12 *Suppose that (G,S,V) is a secure signature scheme, and that (G',S',V') is a secure one-time signature scheme. Then (G'',S'',V''), as defined in Construction 6.4.11 is a secure signature scheme.*

We comment that the proposition holds even if (G,S,V) is only secure against attackers that select queries according to the distribution $G'_2(1^n)$. Furthermore, (G,S,V) need only be ℓ-restricted, for some suitable function $\ell : \mathbb{N} \to \mathbb{N}$.

Proof Sketch: Consider an adversary A'' attacking the scheme (G'',S'',V''). We may ignore the case in which two queries of A'' are answered by triplets containing the same one-time verification-key v' (since if this event occurs with non-negligible probability then the one-time scheme (G',S',V') cannot be secure). We consider two cases regarding the relation of the values of the one-time verification-key in the signatures provided by S''_s and its value in the signature forged by A''.

1. In case, *for some* i, the one-time verification-key v' contained in the forged message equals the one-time verification-key $v^{(i)}$ contained in the answer to the i^{th} query, we derive violation to the security of the one-time scheme (G',S',V').

   Specifically, consider an adversary A' that on input a verification-key v' for the one-time scheme (G',S',V'), generates $(s,v) \leftarrow G(1^n)$ at random, selects i at random (among polynomially many possibilities), invokes A'' on input v, and answers its queries as follows. The i^{th} query of A'', denoted $\alpha^{(i)}$, is answered by making the only query to $S'_{s'}$, obtaining $\beta' = S'_{s'}(\alpha^{(i)})$, and returning $(S_s(v'),v',\beta')$ to A''. (Note that A' holds s.) Each other query of A'', denoted $\alpha^{(j)}$, is answered by invoking G' to obtain $(s^{(j)},v^{(j)}) \leftarrow G'(1^n)$, and returning $(S_s(v^{(j)}),v^{(j)},S'_{s^{(j)}}(\alpha^{(j)})$ to A''. If A'' answers with a forged signature and v' is the verification-key contained in it, then A' obtains a forged signature relative to the one-time scheme (G',S',V') (i.e., a signature to a message different from $\alpha^{(i)}$, which is valid w.r.t the verification-key v'). Conditioned on the case hypothesis and a forgery event, the second event (i.e., v' is the verification-key contained in the forged signature) occurs with $1/\text{poly}(n)$ probability. (Note that indeed A' made a single query to $S'_{s'}$, and that the distribution seen by A'' is exactly as in an actual attack on (G'',S'',V'').)

2. In case, *for all* i, the one-time verification-key v' contained in the forged message is different from the one-time verification-key $v^{(i)}$ contained in the answer to the i^{th} query, we derive violation to the security of the scheme (G,S,V).

   Specifically, consider an adversary A that on input a verification-key v for the scheme (G,S,V), invokes A'' on input v, and answers its queries as follows. To answer the j^{th} query of A'', denoted $\alpha^{(j)}$, algorithm A invokes G' to obtain $(s^{(j)},v^{(j)}) \leftarrow G'(1^n)$, queries S_s for a signature to $v^{(j)}$, and returns $(S_s(v^{(j)}),v^{(j)},S'_{s^{(j)}}(\alpha^{(j)})$ to A''. When A'' answers with a forged signature and $v' \notin \{v^{(j)} : j = 1, ..., \text{poly}(n)\}$ is the one-time verification-key contained in it, A obtains a forged signature relative to the scheme (G, S, V) (i.e., a signature to a string v' different from all $v^{(j)}$'s, which is valid w.r.t the verification-key v). (Note again that the distribution seen by A'' is exactly as in an actual attack on (G'', S'', V'').)[9]

Thus in both cases we derive a contradiction to some hypothesis, and the proposition follows. □

6.4.2.2 Authentication–trees

The refreshing paradigm by itself (i.e., as employed in Construction 6.4.11) does not seem to be enough for establishing Theorem 6.4.9. Recall that our aim is to construct a general signature scheme based on a one-time signature scheme. The refreshing paradigm suggests to use a fresh instance of a one-time signature scheme in order to sign the actual document; however, whenever we do so (as in Construction 6.4.11), we must autheticate this fresh instance relative to the single verification-key that is public. A straightforward implementation of this scheme (as presented in Construction 6.4.11) calls for many signatures to be signed relative to the single verification-key that is public, and so a one-time signature scheme cannot be used (for this purpose). Instead, a more sophisticated method of authentication is required.

Let us try to sketch the basic idea underlying the new authentication method. The idea is to use the public verification-key (of a one-time signature scheme) in order to authenticate several (e.g., two) fresh instances, use each of these instances to authenticate several fresh instances, and so on. We obtain a tree of fresh instances of the one-time signature, where each internal node authenticates its children. See Figure 6.1 (below). We can now use the leaves of this tree in order to sign actual documents, where each leave is used at most once. We stress that each instance of the one-time signature scheme is used to sign at most one string (i.e., a sequence of verification-keys if the instance resides in an internal node, and an actual document if the instance resides in a leaf).

The above description may leave the reader wondering as to how one actually signs (and verifies signatures) using the suggested signature scheme. We start with a description that does not fit our definition of a signature scheme, because it requires the signer to keep a record of its actions during previous invocations of the signing process.[10] We refer to such a scheme as *memory dependent*.

Definition 6.4.13 (memory-dependent signature schemes):

mechanics: *Item 1 of Definition 6.1.1 stays as it is, and the* initial state (of the signing algorithm) *is defined to equal the output of the key-generator. Item 2 is modified so that the signing algorithm is given a state, denoted γ, as auxiliary input and returns a modified state, denoted δ, as auxiliary*

[9] Furthermore, all queries to S_s are distributed according to $G_2(1^n)$, justifying the comment made just before the proof sketch.

[10] This (memory) requirement will be removed in the next section.

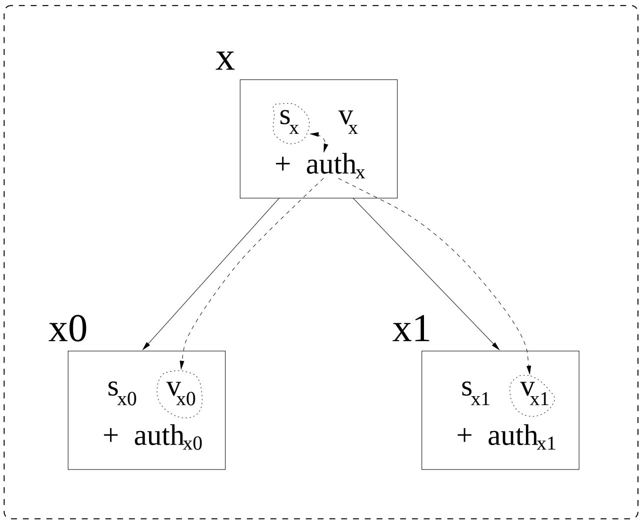

Figure 6.1: A node labeled x authenticates its children, labeled $x0$ and $x1$, respectively. The authentication is via a one-time signature of the text $v_{x0}v_{x1}$ using signing-key s_x.

output. It is required that for every pair (s,v) in the range of $G(1^n)$, and for every $\alpha,\gamma \in \{0,1\}^*$, if $S_s(\alpha,\gamma) = (\beta,\delta)$ then $V_v(\alpha,\beta) = 1$ and $|\delta| \leq |\gamma| + |\alpha| \cdot \mathrm{poly}(n)$.

(That is, the verification algorithm accepts the signature β and the state does not grow by too much.)

security: *The notion of a chosen message attack is modified so that the oracle S_s now maintains a state that it updates in the natural manner; that is, when in state γ and faced with query α, the oracle sets $(\beta,\delta) \leftarrow S_s(\alpha,\gamma)$, returns β and updates its state to δ. The notions of success and security are defined as in Definition 6.1.2, except that they now refer to the modified notion of an attack.*

We note that memory-dependent signature schemes may suffice in many applications of signature schemes. Still, it is preferable to have memoryless (i.e., ordinary) signature schemes. Below we use any one-time signature schemes to construct a memory-dependent signature scheme. The memory requirement will be removed in the next section, so to obtain a (memoryless) signature scheme (as in Definition 6.1.1).

Construction 6.4.14 (a memory-dependent signature scheme): *Let (G,S,V) be a one-time signature scheme. Consider the following memory-dependent signature scheme, (G',S',V'), with $G' = G$. On security parameter n, the scheme uses a full binary tree of depth n. Each of the nodes in this tree is* labeled *by a* *binary string so that the root is labeled by the empty string, denoted λ, and the left* (resp., right) *child of a node labeled by x is labeled by $x0$* (resp., $x1$)*. Below we refer to the current state of the signing process as to a record.*

initiating the scheme: *To initiate the scheme, on security parameter n, we invoke $G(1^n)$ and let $(s,v) \leftarrow G(1^n)$. We record (s,v) as the key-pair associated with the root, and output v as the (public) verification-key.*

In the rest of the description, we denote by (s_x, v_x) the key-pair associated with the node labeled x; thus, $(s_\lambda, v_\lambda) = (s,v)$.

signing with S' using the current record: *Recall that the current recond contains the signing-key $s = s_\lambda$, which is used to produce* auth_λ *(defined below).*

To sign a new document, denoted α, we first allocate an unused leaf. Let $\sigma_1 \cdots \sigma_n$ be the label of this leaf. For example, we may keep a counter of the number of documents signed, and determine $\sigma_1 \cdots \sigma_n$ according to the counter value (e.g., if the counter value is c then we use the c^{th} string in lexicographic order).

Next, for every $i = 1, ..., n$ and every $\tau \in \{0,1\}$, we try to retreive from our record the key-pair associated with the node labeled $\sigma_1 \cdots \sigma_{i-1}\tau$. In case such a pair is not found, we generate it by invoking $G(1^n)$ and store it (i.e., add it to our record) for future use; that is, we let $(s_{\sigma_1 \cdots \sigma_{i-1}\tau}, v_{\sigma_1 \cdots \sigma_{i-1}\tau}) \leftarrow G(1^n)$.

For every $i = 1, ..., n$, we try to retreive from our record a signature to the string $v_{\sigma_1 \cdots \sigma_{i-1}0}\, v_{\sigma_1 \cdots \sigma_{i-1}1}$ relative to the signing-key $s_{\sigma_1 \cdots \sigma_{i-1}}$. In case such a signature is not found, we generate it by invoking $S_{s_{\sigma_1 \cdots \sigma_{i-1}}}$, and store it for future use; that is, we obtain $S_{s_{\sigma_1 \cdots \sigma_{i-1}}}(v_{\sigma_1 \cdots \sigma_{i-1}0}\, v_{\sigma_1 \cdots \sigma_{i-1}1})$. We let

$$\text{auth}_{\sigma_1 \cdots \sigma_{i-1}} \stackrel{\text{def}}{=} \left(v_{\sigma_1 \cdots \sigma_{i-1}0}\,,\, v_{\sigma_1 \cdots \sigma_{i-1}1}\,,\, S_{s_{\sigma_1 \cdots \sigma_{i-1}}}(v_{\sigma_1 \cdots \sigma_{i-1}0}\, v_{\sigma_1 \cdots \sigma_{i-1}1})\right)$$

(Intuitively, via $\text{auth}_{\sigma_1 \cdots \sigma_{i-1}}$ the node labelled $\sigma_1 \cdots \sigma_{i-1}$ authenticates the verification-keys associated with its children.)

Finally, we sign α by invoking $S_{s_{\sigma_1 \cdots \sigma_n}}$, and output

$$(\sigma_1 \cdots \sigma_n, \text{auth}_\lambda, \text{auth}_{\sigma_1}, ..., \text{auth}_{\sigma_1 \cdots \sigma_{n-1}}, S_{s_{\sigma_1 \cdots \sigma_n}}(\alpha))$$

verification with V': *On input a verification-key v, a document α, and an alledge signature β we accept if and only if the following conditions hold:*

1. *β has the form*

$$(\sigma_1 \cdots \sigma_n, (v_{0,0}, v_{0,1}, \beta_0), (v_{1,0}, v_{1,1}, \beta_1), ..., (v_{n-1,0}, v_{n-1,1}, \beta_{n-1}), \beta_n)$$

where the σ_i's are bits and all other symbols represent strings.

(Jumping ahead, we mention that $v_{i,\tau}$ is supposed to equal $v_{\sigma_1 \cdots \sigma_{i-1}\tau}$, the verification-key associated by the signing process with the node labeled $\sigma_1 \cdots \sigma_{i-1}\tau$. In particular, v_{i,σ_i} is supposed to equal $v_{\sigma_1 \cdots \sigma_i}$.)

2. $V_v(v_{0,0}v_{0,1}, \beta_0) = 1$.
(That is, the public-key (i.e., v) authenticates the two strings $v_{0,0}$ and $v_{0,1}$ claimed to correspond to the instances of the one-time signature scheme associated with the nodes labeled 0 and 1, respectively.)
3. *For $i = 1, ..., n-1$, it holds that* $V_{v_{i-1,\sigma_i}}(v_{i,0}v_{i,1}, \beta_i) = 1$.
(That is, the verification-key v_{i-1,σ_i}, which is already beleived to be authentic and supposedly corresponds to the instance of the one-time signature scheme associated with the node labeled $\sigma_1 \cdots \sigma_i$, authenticates the two strings $v_{i,0}$ and $v_{i,1}$ that are supposed to correspond to the instances of the one-time signature scheme associated with the nodes labeled $\sigma_1 \cdots \sigma_i 0$ and $\sigma_1 \cdots \sigma_i 1$, respectively.)
4. $V_{v_{n-1,\sigma_n}}(\alpha, \beta_n) = 1$.
(That is, the verification-key v_{n-1,σ_n}, which is already beleived to be authentic, authenticates the actual document α.)

Regarding the verification algorithm, note that Conditions 2 and 3 establish that $v_{i,\sigma_{i+1}}$ is authentic (i.e., equals $v_{\sigma_1 \cdots \sigma_i, \sigma_{i+1}}$). That is, $v = v_\lambda$ authenticates v_{σ_1}, which authenticates $v_{\sigma_1 \sigma_2}$, and so on up-to $v_{\sigma_1 \cdots \sigma_n}$. The fact that the $v_{i,1-\sigma_{i+1}}$'s are proven to be authentic (i.e., equal the $v_{\sigma_1 \cdots \sigma_i, 1-\sigma_{i+1}}$'s) is not really useful. This is merely an artifact of the fact that $s_{\sigma_1 \cdots \sigma_i}$ can be (securely) used to produce a single signature, during the entire operation of the memory-dependent signature scheme. In the currently (constructed) S'_s-signature we may not care about the authenticity of some $v_{\sigma_1 \cdots \sigma_i, 1-\sigma_{i+1}}$, but we may care about it in some other S'_s-signature. For example, if we use the leaf labeled 0^n to sign the first document and the leaf labeled $0^{n-1}1$ to sign the second, then in the first S'_s-signature we only care about the authenticity of v_{0^n}, whereas in the second S'_s-signature we care about the authenticity of $v_{0^{n-1}1}$.

Proposition 6.4.15 *If (G, S, V) is a secure one-time signature scheme then Construction 6.4.14 constitutes a secure memory-dependent signature scheme.*

Proof: Recall that a S'_{s_λ}-signature to a document α has the form

$$(\sigma_1 \cdots \sigma_n, \mathrm{auth}_\lambda, \mathrm{auth}_{\sigma_1}, ..., \mathrm{auth}_{\sigma_1 \cdots \sigma_{n-1}}, S_{s_{\sigma_1 \cdots \sigma_n}}(\alpha)) \tag{6.3}$$

where the auth_x's, v_x's and s_x's satisfy

$$\mathrm{auth}_x = (v_{x0}\,,\, v_{x1}\,,\, S_{s_x}(v_{x0}\, v_{x1})) \tag{6.4}$$

(See Figure 6.1.) In this case we say that this S'_s-signature uses the leaf labeled $\sigma_1 \cdots \sigma_n$. For every $i = 1, ..., n$, we call the sequence $(\mathrm{auth}_\lambda, \mathrm{auth}_{\sigma_1}, ..., \mathrm{auth}_{\sigma_1 \cdots \sigma_{i-1}})$ an authentication path for $v_{\sigma_1 \cdots \sigma_i}$. Note that the above sequence is also an authentication path for $v_{\sigma_1 \cdots \sigma_{i-1} \overline{\sigma}_i}$, where $\overline{\sigma} = 1 - \sigma$. Thus, a valid S'_s-signature to a document α consists of an n-bit string $\sigma_1 \cdots \sigma_n$, authentication paths for each $v_{\sigma_1 \cdots \sigma_i}$ $(i = 1, ..., n)$, and a signature to α with respect to the one-time scheme (G, S, V) using the signing-key $s_{\sigma_1 \cdots \sigma_n}$.

Intuitively, forging an S'_s-signature requires either using an authentication path supplied by the signer (i.e., supplied by S'_s as part of an answer to a query) or producing an authentication path different from all paths supplied by the signer. In both cases, we reach a contradiction to the security of the one-time signature scheme (G, S, V). Specifically, in the first case, the forged S'_s-signature contains a signature relative to (G, S, V) using the signing-key $s_{\sigma_1 \cdots \sigma_n}$. The latter $S_{s_{\sigma_1 \cdots \sigma_n}}$-signature is verifiable using the verification-key $v_{\sigma_1 \cdots \sigma_n}$, which is authentic by the case hypothesis. This yields forgery with respect to the instance of the one-time signature scheme associated with the leaf labeled $\sigma_1 \cdots \sigma_n$ (since the document S'_s-signed by the forger must be different from all S'_s-signed documents, and thus the forged document is different from all strings to which a one-time signature was applied).[11] We now turn to the second case (i.e., forgery with respect to (G', S', V') is obtained by producing an authentication path different from all paths supplied by the signer). In this case there must exists an $i \in \{1, ..., n\}$ and an i-bit long string $\sigma_1 \cdots \sigma_i$ so that $\text{auth}_\lambda, ..., \text{auth}_{\sigma_1 \cdots \sigma_{i-1}}$ is the shortest prefix of the authentication path produced by the forger that is NOT a prefix of any authentication path supplied by the signer. (Note that $i > 0$ must hold, since empty sequences are equal, whereas $i \leq n$ by the case hypothesis.) In this case $\text{auth}_{\sigma_1 \cdots \sigma_{i-1}}$, which contains a signature relative to (G, S, V) using the signing-key $s_{\sigma_1 \cdots \sigma_{i-1}}$, is produced by the forger. The latter signature is verifiable using the verification-key $v_{\sigma_1 \cdots \sigma_{i-1}}$, which is authentic by the minimality of i. Furthermore, by definition of i, the latter signature is to a string different from the string to which the S'_s-signer has applied $S_{s_{\sigma_1 \cdots \sigma_{i-1}}}$. This yields forgery with respect to the instance of the one-time signature scheme associated with the node labeled $\sigma_1 \cdots \sigma_{i-1}$.

The actual proof is by a reducibility argument. Given an adversary A' attacking the complex scheme (G', S', V'), we construct an adversary A that attacks the one-time signature scheme, (G, S, V). In particular, the adversary A will use its oracle access S_s in order to emulate the memory-dependent signing oracle for A'. Recall that the adversary A can make at most one query to its S_s-oracle. Below is a detailed description of the adversary A. Since we care only about probabilistic polynomial-time adversaries, we may assume that A' makes at most $t = \text{poly}(n)$ many queries, where n is the security parameter.

The construction of adversary A: Suppose that (s, v) is in the range of $G(1^n)$. On input v and one-query oracle access to S_s, adversary A proceeds as follows:

1. *Initial choice*: A uniformly selects $j \in \{1, ..., (2n + 1) \cdot t\}$.

   (The integer j specifies an instance of (G, S, V) produced during the attack of A'. This instance will be attacked by A. Note that since $2n+1$ instances of (G, S, V) are refered to in each signature relative to (G', S', V'), the

[11] Note that what matter is merely that the document S'_s-signed by the forger is different from the (single) document to which $S_{s_{\sigma_1 \cdots \sigma_n}}$ was applied by the S'_s-signer, in case $S_{s_{\sigma_1 \cdots \sigma_n}}$ was ever applied by the S'_s-signer.

quantity $(2n+1)\cdot t$ upper bounds the total number of instances of (G, S, V) that appear in some oracle response. This upper bound is not tight.)

2. *Invoking A'*: If $j = 1$ then A sets $v_\lambda = v$ and invokes A' on input v. In this case A does not know s_λ, which is defined to equal s, but can obtain a *single* signature relative to it by making a (single) query to oracle S_s.

   Otherwise (i.e., $j > 1$), machine A invokes G, obtains $(s', v') \leftarrow G(1^n)$, sets $(s_\lambda, v_\lambda) = (s', v')$ and invokes A' on input v'. We stress that in this case A knows s_λ.

   In fact, in both case, A' is invoked on input v_λ. Also, in both cases, the one-time instance associated with the root (i.e., the node labeled λ) is called the first instance.

3. *Emulating the signing oracle for A'*: A emulates the memory-dependent signing oracle for A'. The emulation is analogous to the operation of the signing procedure as specified in Construction 6.4.14. The only execption refer to the j^{th} instance of (G, S, V) that occurs in the memory-dependent signing process. Here, A uses the verification key v, and if an S_s-signature needs to be produced then A queries S_s for it. We stress that at most one signature needs ever be produced with respect to each instance of (G, S, V) that occurs in the memory-dependent signing process, and therefore S_s is queried at most once. Details follow.

   A maintains a recond of all key-pairs and one-time signatures it has generated and/or obtained from S_s. When A is asked to supply a signature to a new document, denoted α, it proceeds as follows:

   (a) A allocates a new leaf-label, denoted $\sigma_1 \cdots \sigma_n$, exactly as done by the signing process.

   (b) For every $i = 1, ..., n$ and every $\tau \in \{0,1\}$, machine A tries to retreive from its record the one-time instance associated with the node labeled $\sigma_1 \cdots \sigma_{i-1}\tau$. If such an instance does not exist in the record (i.e., the one-time instance associated with the node labeled $\sigma_1 \cdots \sigma_{i-1}\tau$ did not appear so far) then A distinguishes two cases:

      i. If the record so far contains exactly $j-1$ one-time instances (i.e., the current instance is the j^{th} one to be encountered) then A sets $v_{\sigma_1 \cdots \sigma_{i-1}\tau} \leftarrow v$, and adds it to its record. In this case, A does not know $s_{\sigma_1 \cdots \sigma_{i-1}\tau}$, which is defined to equal s, but can obtain a *single* signature relative to it by making a (single) query to oracle S_s.
      From this point on, the one-time instance associated with the node labeled $\sigma_1 \cdots \sigma_{i-1}\tau$ will be called the j^{th} instance.

      ii. Otherwise (i.e., the current instance is NOT the j^{th} one to be encountered), A acts as the signing process: It invokes $G(1^n)$, obtains $(s_{\sigma_1 \cdots \sigma_{i-1}\tau}, v_{\sigma_1 \cdots \sigma_{i-1}\tau}) \leftarrow G(1^n)$, and adds it to the record.

(Note that in this case A knows $s_{\sigma_1\cdots\sigma_{i-1}\tau}$, and can generate by itself signatures relative to it.)
The one-time instance just generated is given the next serial number. That is, the one-time instance associated with the node labeled $\sigma_1\cdots\sigma_{i-1}\tau$ will be called the $k^{\rm th}$ instance if the current record (i.e., after the generation of the one-time key-pair associated with the node labeled $\sigma_1\cdots\sigma_{i-1}\tau$) contains exactly k instances.

(c) For every $i=1,...,n$, machine A tries to retreive from its record a (one-time) signature to the string $v_{\sigma_1\cdots\sigma_{i-1}0}\,v_{\sigma_1\cdots\sigma_{i-1}1}$, relative to the signing-key $s_{\sigma_1\cdots\sigma_{i-1}}$. If such a signature does not exist in the record then A distinguishes two cases:

  i. If the one-time signature instance associated with the node labeled $\sigma_1\cdots\sigma_{i-1}$ is the $j^{\rm th}$ such instance then A obtains the one-time signature $S_{s_{\sigma_1\cdots\sigma_{i-1}}}(v_{\sigma_1\cdots\sigma_{i-1}0}\,v_{\sigma_1\cdots\sigma_{i-1}1})$ by querying S_s, and adds this signature to the record.
  Note that by the previous steps (i.e., Step 3(b)i as well as Step 2), s is indentified with $s_{\sigma_1\cdots\sigma_{i-1}}$, and that the instance associated with a node labeled $\sigma_1\cdots\sigma_{i-1}$ is only used to produce a single signature; that is, to the string $v_{\sigma_1\cdots\sigma_{i-1}0}\,v_{\sigma_1\cdots\sigma_{i-1}1}$. Thus, in this case, A queries S_s at most once.
  We stress that the above makes crucial use of the fact that, for every τ, the verification-key associated with the node labeled $\sigma_1\cdots\sigma_{i-1}\tau$ is identical in all executions of the current step, regardless of whether it is generated in Step 3(b)ii or fixed to equal v (in Step 3(b)i). This fact guarantees that A only needs a single signature relative to the instance associated with a node labeled $\sigma_1\cdots\sigma_{i-1}$, and thus queries S_s at most once. (The validity of this fact is the most important place in which we rely on the memory-dependence of our signature scheme.)[12]

  ii. Otherwise (i.e., the one-time signature instance associated with the node labeled $\sigma_1\cdots\sigma_{i-1}$ is NOT the $j^{\rm th}$ such instance), A acts as the signing process: It invokes $S_{s_{\sigma_1\cdots\sigma_{i-1}}}$, obtains the one-time signature $S_{s_{\sigma_1\cdots\sigma_{i-1}}}(v_{\sigma_1\cdots\sigma_{i-1}0}\,v_{\sigma_1\cdots\sigma_{i-1}1})v_{\sigma_1\cdots\sigma_{i-1}\tau})$, and adds it to the record. (Note that in this case A knows $s_{\sigma_1\cdots\sigma_{i-1}}$, and can generate by itself signatures relative to it.)

Thus, A obtains ${\rm auth}_{\sigma_1\cdots\sigma_{i-1}}$.

(d) Machine A now obtains a one-time signature of α relative to $S_{s_{\sigma_1\cdots\sigma_n}}$. (Recall that since A' never makes the same query twice,[13] we need

[12] In contrast, the use of a counter for determining a new leaf can be easily avoided, by selecting a leaf at random.

[13] This assertion can be justified, without loss of generality. Otherwise, we may modify A' so that retreives from its own memory the answer to a query that it wishes to ask for the second time.

to generate at most one signature relative to the one-time instance $S_{s_{\sigma_1\cdots\sigma_n}}$.) This is done analogously to the previous step (i.e., Step 3c). Specifically:

i. If the one-time signature instance associated with the leaf labeled $\sigma_1\cdots\sigma_n$ is the $j^{\rm th}$ instance (associated with any node) then A obtains the one-time signature $S_{s_{\sigma_1\cdots\sigma_n}}(\alpha)$ by querying S_s.
Note that, in this case, s is indentified with $s_{\sigma_1\cdots\sigma_n}$, and that an instance associated with a leaf is only used to produce a single signature. Thus, also in this case (which is disjoint of Case 3(c)i), A queries S_s at most once.

ii. Otherwise (i.e., the one-time signature instance associated with the node labeled $\sigma_1\cdots\sigma_n$ is NOT the $j^{\rm th}$ instance), A acts as the signing process: It invokes $S_{s_{\sigma_1\cdots\sigma_n}}$, obtains the one-time signature $S_{s_{\sigma_1\cdots\sigma_n}}(\alpha)$, and adds it to the record. (Again, in this case A knows $s_{\sigma_1\cdots\sigma_n}$, and can generate by itself signatures relative to it.)

Thus, A obtains $\beta_n = S_{s_{\sigma_1\cdots\sigma_n}}(\alpha)$.

(e) Finally, A answers the query α with

$$(\sigma_1\cdots\sigma_n, \mathrm{auth}_\lambda, \mathrm{auth}_{\sigma_1}, ..., \mathrm{auth}_{\sigma_1\cdots\sigma_{n-1}}, \beta_n)$$

4. *Using the output of A'*: When A' halts with output (α',β'), machine A checks whether this is a valid document-signature pair with respect to V'_{v_λ} and whether the document α' did not appear as a query of A'. If both conditions hold then A tries to obtain forgery with respect to S_s. To explain how this is done, we need to take a closer look at the valid document-signature pair, (α',β'), output by A'. Specifically, suppose that β' has the form

$$(\sigma'_1\cdots\sigma'_n, (v'_{0,0}, v'_{0,1}, \beta'_0), (v'_{1,0}, v'_{1,1}, \beta'_1), ..., (v'_{n-1,0}, v'_{n-1,1}, \beta'_{n-1}), \beta'_n)$$

and that the various components satisfy all conditions stated in the verification procedure. (In particular, the sequence $(v'_{0,0}, v'_{0,1}, \beta'_0), ..., (v'_{n-1,0}, v'_{n-1,1}, \beta'_{n-1})$ is the authentication path (for v'_{n-1,σ'_n}) output by A'.) Let i be *maximal* so that for *some* $\beta_0,...,\beta_{i-1}$ (which may but need not equal $\beta'_0,...,\beta'_{i-1}$) the sequence $(v'_{0,0}, v'_{0,1}, \beta_0), ..., (v'_{i-1,0}, v'_{i-1,1}, \beta_{i-1})$ is a prefix of *some* authentication path (for some $v_{\sigma'_1\cdots\sigma'_i\sigma_{i+1}\cdots\sigma_n}$) supplied to A' by A. Note that $i\in\{0,...,n\}$, where $i=0$ means that $(v'_{0,0}, v'_{0,1})$ differs from (v_0,v_1), and $i=n$ means that the sequence $((v'_{0,0}, v'_{0,1}), ..., (v'_{n-1,0}, v'_{n-1,1}))$ equals the sequence $((v_0,v_1), ..., (v_{\sigma'_1\cdots\sigma'_{n-1}0}, v_{\sigma'_1\cdots\sigma'_{n-1}1}))$.

Recall that the $v'_{k,\tau}$s are strings included in the output of A', and that the v_xs are verification-keys as recorded by A. In general, the sequence $((v'_{0,0}, v'_{0,1}), ..., (v'_{i-1,0}, v'_{i-1,1}))$ equals the sequence $((v_0,v_1), ..., (v_{\sigma'_1\cdots\sigma'_{i-1}0}, v_{\sigma'_1\cdots\sigma'_{i-1}1}))$. In particular, for $i\geq 1$, it holds that $v'_{i-1,\sigma'_{i-1}} = v_{\sigma'_1\cdots\sigma'_i}$, whereas for $i=0$

we shall only refer to v_λ (which is the verification-key attacked by A'). In both cases, the output of A' contains a one-time signature relative to $v_{\sigma'_1 \cdots \sigma'_i}$, and this signature is to a string different from the only (possible) one to which a signature was supplied to A' by A. Analogously to the motivating discussion above, we distinguish the cases $i = n$ and $i < n$:

(a) In case $i = n$, the output of A' contains the (one-time) signature β'_n that satisfies $V_{v_{\sigma'_1 \cdots \sigma'_n}}(\alpha', \beta'_n) = 1$. Furthermore, α' is different from the (possibly) only document to which $S_{s_{\sigma'_1 \cdots \sigma'_n}}$ was applied during the emulation of the S'-signer by A, since by our hypothesis the document α' did not appear as a query of A'. (Recall that, by the construction of A, instances of the one-time signature scheme associated with leaves are only applied to the queries of A'.)

(b) In case $i < n$, the output of A' contains the (one-time) signature β'_i that satisfies $V_{v_{\sigma'_1 \cdots \sigma'_i}}(v'_{i,0}v'_{i,1}, \beta'_i) = 1$. Furthermore, $v'_{i,0}v'_{i,1}$ is different from $v_{\sigma'_1 \cdots \sigma'_i 0}\, v_{\sigma'_1 \cdots \sigma'_i 0}$, which is the (possibly) only string to which $S_{s_{\sigma'_1 \cdots \sigma'_i}}$ was applied during the emulation of the S'-signer by A, where the last assertion is due to the maximality of i (and the construction of A).

Thus, in both cases, A obtains from A' a valid (one-time) signature relative to the (one-time) instance associated with the node labeled $\sigma'_1 \cdots \sigma'_i$. Furthermore, in both cases, this (one-time) signature is to a string that did not appear in the record of A. The question is whether the instance associated with the node labeled $\sigma'_1 \cdots \sigma'_i$ is the j^{th} instance, for which A set $v = v_{\sigma'_1 \cdots \sigma'_i}$. In case the answer is yes, A obtains forgery with respect to the (one-time) verification-key v (which it attacks).

In view of the above discussion, A acts as follows. It determines i as in the discussion, and checks whether $v = v_{\sigma'_1 \cdots \sigma'_i}$ (almost equivalently, whether the j^{th} instance is the one associated with the node labeled $\sigma'_1 \cdots \sigma'_i$). In case $i = n$, machine A outputs the string-signature pair (α', β'_n), otherwise (i.e., $i < n$) it outputs the string-signature pair $(v'_{i,0}v'_{i,1}, \beta'_i)$.

This completes the (admittingly long) description of adversary A. We repeat again some obvious observations regarding this construction. Firstly, A makes at most one query to its (one-time) signature oracle S_s. Secondly, assuming that A' is probabilistic polynomial-time, so is A. Thus, all that remains is to relate the success probability of A (when attacking a random instance of (G, S, V)) to the success probability of A' (when attacking a random instance of (G', S', V')). As usual the main observation is that the view of A', during the emulation (of the memory-dependent signing process) by A, is identically distributed to its view in an actual attack on (G', S', V'). Furthermore, this holds conditioned on any possible fixed value of j (selected in the first step of A). It follows that if A' succeeds to forge signatures in an actual attack on (G', S', V') with probability $\varepsilon'(n)$ then A succeeds to forge signatures with respect to (G, S, V)

with probability at least $\frac{\varepsilon'(n)}{(2n+1)\cdot t}$, where the $(2n+1)\cdot t$ factor is due to the probability that the choice of j is a good one (i.e., so that the j^{th} instance is the one associated with the node labeled $\sigma'_1\cdots\sigma'_i$, where $\sigma'_1\cdots\sigma'_n$ and i are as defined in Step 4).

We conclude that if (G',S',V') can be broken by a probabilistic polynomial-time chosen message attack with non-negligible probability then (G,S,V) can be broken by a probabilistic polynomial-time single-message attack with non-negligible probability, in contradiction to the proposition's hypothesis. The proposition follows. $\blacksquare$

6.4.2.3 The actual construction

In this section, we remove the memory-dependency of Construction 6.4.14, and obtain an ordinary (rather than memory-dependent) signature scheme. Towards this end, we use pseudorandom functions (as defined in Definition 3.6.4). The basic idea is that the record maintained in Construction 6.4.14 can be determined (on-the-fly) by an application of a pseudorandom function to certain strings. For example, instead of generating and storing an instance of a (one-time) signature scheme for each node we encounter, we can determine the randomness for the key-generation algorithm as a function of the label of that node. Thus, there is no need to store the key-pair generated, since if we ever need it again then re-generating it (in the very same way) will yield exectly the same result. The same idea applies also to the generation of (one-time) signatures. In fact, the construction is simplified, since we need not check whether or not we are generating an object for the first time.

For simplicity, let us assume that on security parameter n both the key-generation and signing algorithms (of the one-time signature scheme (G,S,V)) use exactly n internal coin tosses. (This assumption can be justified by using pseudorandom generators, which exist anyhow under the assumptions used here.) For $r\in\{0,1\}^n$, we denote by $G(1^n,r)$ the output of G on input 1^n and internal coin-tosses r. Likwise, for $r\in\{0,1\}^n$, we denote by $S_s(\alpha,r)$ the output of S, on input a signing-key s and a document α, when using internal coin-tosses r. For simplicity, we shall be actually using generalized pseudorandom functions as in Definition 3.6.12 (rather than pseudorandom functions as defined in Definition 3.6.4).[14] Furthermore, for simplicity, we shall consider applications of such pseudorandom functions to sequences of characters containing $\{0,1\}$ as well as a few additional special characters.

Construction 6.4.16 (Removing the memory requirement from Construction 6.4.14): *Let (G,S,V) be a one-time signature scheme, and $\{f_r:\{0,1\}^*\to\{0,1\}^{|r|}\}_{r\in\{0,1\}^*}$ be a generalized pseudorandom function ensemble as in Definition 3.6.12. Con-*

[14] We shall make comments regarding the minor changes required in order to use ordinary pseudorandom functions. The first comment is that we shall consider an encoding of strings of length up-to $n+2$ by strings of length $n+3$ (e.g., for $i\le n+2$, the string $x\in\{0,1\}^i$ is encoded by $x10^{n+2-i}$).

sider the following signature scheme, (G', S', V'), which refers to a full binary tree of depth n as in Construction 6.4.14.

key-generation algorithm G': *On input 1^n, algorithm G' obtains $(s,v) \leftarrow G(1^n)$ and selects uniformly $r \in \{0,1\}^n$. Algorithm G' outputs the pair $((r,s),v)$, where (r,s) is the signing-key and v is the verification-key.*[15]

signing algorithm S': *On input a signing-key (r,s) and a document α, the algorithm proceeds as follows.*

1. *It selects uniformly $\sigma_1 \cdots \sigma_n \in \{0,1\}^n$.*

   (Algorithm S' will use the leaf labeled $\sigma_1 \cdots \sigma_n \in \{0,1\}^n$ to sign the current document. Indeed, with exponentially-vanishing probability the same leaf may be used to sign two different documents, and this will lead to forgey (but only with negligible probability).)

   (Alternatively, to obtain a deterministic signing algorithm, one may set $\sigma_1 \cdots \sigma_n \leftarrow f_r(\texttt{select-leaf}, \alpha)$, where `select-leaf` is a special character.)[16]

2. *Next, for every $i = 1, ..., n$ and every $\tau \in \{0,1\}$, the algorithm invokes G and sets*

$$(s_{\sigma_1 \cdots \sigma_{i-1}\tau}, v_{\sigma_1 \cdots \sigma_{i-1}\tau}) \leftarrow G(1^n, f_r(\texttt{key-gen}, \sigma_1 \cdots \sigma_{i-1}\tau))$$

   where `key-gen` *is a special character.*[17]

3. *For every $i = 1, ..., n$, the algorithm invokes $S_{s_{\sigma_1 \cdots \sigma_{i-1}}}$ and sets*

$$\begin{aligned} \mathrm{auth}_{\sigma_1 \cdots \sigma_{i-1}} \stackrel{\mathrm{def}}{=} \big(& v_{\sigma_1 \cdots \sigma_{i-1}0} \, , \, v_{\sigma_1 \cdots \sigma_{i-1}1} \, , \\ & S_{s_{\sigma_1 \cdots \sigma_{i-1}}}(v_{\sigma_1 \cdots \sigma_{i-1}0} \, v_{\sigma_1 \cdots \sigma_{i-1}1}, f_r(\texttt{sign}, \sigma_1 \cdots \sigma_{i-1})) \big) \end{aligned}$$

   where `sign` *is a special character.*[18]

4. *Finally, the algorithm invokes $S_{s_{\sigma_1 \cdots \sigma_n}}$ and outputs*[19]

$$(\sigma_1 \cdots \sigma_n, \mathrm{auth}_\lambda, \mathrm{auth}_{\sigma_1}, ..., \mathrm{auth}_{\sigma_1 \cdots \sigma_{n-1}}, S_{s_{\sigma_1 \cdots \sigma_n}}(\alpha, f_r(\texttt{sign}, \sigma_1 \cdots \sigma_n)))$$

---

[15] In case we use ordinary pseudorandom functions, rather than generalized ones, we select r uniformly in $\{0,1\}^{n+3}$ so that $f_r : \{0,1\}^{n+3} \to \{0,1\}^{n+3}$. Actually, we shall be using the function $f_r : \{0,1\}^{n+3} \to \{0,1\}^n$ derived from the above by droping the last 3 bits of the function value.

[16] In case we use ordinary pseudorandom functions, rather than generalized ones, this alternative can be (directly) implemented only if it is guaranteed that $|\alpha| \leq n$. In such a case, we apply the f_r to the $(n+3)$-bit encoding of 00α.

[17] In case we use ordinary pseudorandom functions, rather than generalized ones, the argument to f_r is the $(n+3)$-bit encoding of $10\sigma_1 \cdots \sigma_{i-1}\tau$.

[18] In case we use ordinary pseudorandom functions, rather than generalized ones, the argument to f_r is the $(n+3)$-bit encoding of $11\sigma_1 \cdots \sigma_{i-1}$.

[19] In case we use ordinary pseudorandom functions, rather than generalized ones, the argument to f_r is the $(n+3)$-bit encoding of $11\sigma_1 \cdots \sigma_n$.

verification algorithm V': *On input a verification-key v, a document α, and an alledge signature β algorithm V' behaves exactly as in Construction 6.4.14. Specifically, assuming that β has the form*

$$(\sigma_1 \cdots \sigma_n, (v_{0,0}, v_{0,1}, \beta_0), (v_{1,0}, v_{1,1}, \beta_1), ..., (v_{n-1,0}, v_{n-1,1}, \beta_{n-1}), \beta_n)$$

algorithm V' accepts if and only if the following three conditions hold:

- $V_v(v_{0,0}v_{0,1}, \beta_0) = 1$.
- *For $i = 1, ..., n-1$, it holds that $V_{v_{i-1,\sigma_i}}(v_{i,0}v_{i,1}, \beta_i) = 1$.*
- $V_{v_{n-1,\sigma_n}}(\alpha, \beta_n) = 1$.

Proposition 6.4.17 *If (G, S, V) is a secure one-time signature scheme and $\{f_r : \{0,1\}^* \to \{0,1\}^{|r|}\}_{r \in \{0,1\}^*}$ is a generalized pseudorandom function ensemble then Construction 6.4.16 constitutes a secure signature scheme.*

Proof: Following the general methodology suggested in Section 3.6.3, we consider an *ideal version* of Construction 6.4.16 in which a truely random function is used (rather than a pseudorandom one). The ideal version is almost identical to Construction 6.4.14, with the only difference being the way in which $\sigma_1 \cdots \sigma_n$ is selected. Specifically, applying a random function to determine (one-time) key-pairs and (one-time) signatures, is equivalent to generating these keys and signatures at random (on-the-fly) and re-using the strored values whenever necessary. Regarding the way in which $\sigma_1 \cdots \sigma_n$ is selected, observe that the proof of Proposition 6.4.15 is oblivious of this way except for the assumption that the same leaf is never used to sign two different documents. However, the probability that the same leaf is used twice by the (memoryless) signing algorithm, when serving polynomially-many signing requests, is exponentially-vanishing and thus can be ignored in our analysis. We conclude that the ideal scheme (in which a truely random function is used instead of f_r) is secure. It follows that also the actual signature scheme (as in Construction 6.4.16) is secure, or else one can efficiently distinguish a pseudorandom function from a truely random one (which is impossible). Details follow.

Assume towards the contradiction that there exists a probabilistic polynomial-time adversary A' that succeeds to forge signatures with respect to (G', S', V') with non-negligible probability, but succeeeds only with negligible probability when attacking the ideal scheme. We construct a distinguisher D that on input 1^n and oracle access to $f : \{0,1\}^* \to \{0,1\}^n$ bevaves as follows. Machine D generates $((r, s), v) \leftarrow G'(1^n)$, and invokes A' on input v. Machine D answers the queries of A' by running the signing process, using the signing-key (r, s), with the exception that it replaces the values $f_r(x)$ by $f(x)$. That is, whenever the signing process calls for the computation of the value of the function f_r on some string x, machine D queries its oracle (i.e., f) on the string x, and uses the respond $f(x)$ instead of $f_r(x)$. When A' outputs an alledge signature to a new documment, machine M evaluates whether the signature is valid (with respect to V_v) and output 1 if and only if A' has indeed succeeded (i.e., the signature is valid). Observe that if D is given oracle access to a truely random function then the emulated A' attacks the ideal scheme, whereas if D is given oracle access to a pseudorandom function f_r then the emulated A' attacks the real scheme. It follows that D distinguishes the two cases, in contradiction to the pseudorandomness of the ensemble $\{f_r\}$. ■

6.4.2.4 Conclusions and comments

Theorem 6.4.9 follows by combining Proposition 6.4.17 with the fact that the existence of secure one-time signature schemes implies the existence of one-way functions (see Exercise 6), which in turn imply the existence of (generalized) pseudorandom functions. Recall that combining Theorem 6.4.9 and Corollary 6.4.8, we obtain Corollary 6.4.10 that states that *the existence of collision-free hashing collections implies the existence of secure signature schemes.*

We comment that Constructions 6.4.14 and 6.4.16 can be generalized as follows. Rather than using a depth n full binary tree, one can use any tree that has a super-polynomial (in n) number of leaves, provided that one can enumerate the leaves (resp., uniformly select a leaf), and generate the path from the root to a given leaf. We consider a few possibilities:

- For any $d:\mathbb{N}\rightarrow\mathbb{N}$ bounded by a polynomial in n (e.g., $d\equiv 2$ or $d(n)=n$ are indeed "extreme" cases), we may consider a full $d(n)$-ary tree of depth $e(n)$ so that $d(n)^{e(n)}$ is greater than any polynomial in n. The above choice of pramaters (i.e., $d\equiv 2$ and $e(n)=n$) is probably the simplest one. Note that the length of the signatures in a generalized construction is linear in $d(n)\cdot e(n)$, the number of applications of the underlying one-time signature scheme (per each general signature) is linear in $e(n)$, and that in internal nodes the one-time signature scheme is applied to string of length linear in $d(n)$. Thus, the choice of parameters may depend on the underlying one-time signature scheme. In fact, $d\equiv 2$ seems a reasonable generic choice, but in some special cases (see Section 6.5.2) one may prefer to use larger $d:\mathbb{N}\rightarrow\mathbb{N}$.

- For the memory-dependent construction, it may be preferable to use unbalanced trees (i.e., having leaves at various levels). The advantage is that if one utilizes first the leaves closer to the root then one can obtain a saving on the cost of signing the first documents.

  For example, consider using a ternary tree of super-logarithmic depth (i.e., $d\equiv 3$ and $e(n)=\omega(\log n)$) in which each internal node of level $i\in\{0,1,...,e(n)-2\}$ has a two children that are internal nodes and a single child that is a leaf (and the internal nodes of level $e(n)-1$ have only leaves as children). Thus, for $i\geq 1$, there are 3^{i-1} leaves at level i. If we use all leaves of level i before using any leave of level $i+1$ then the length of the $j^{\rm th}$ signature in this scheme is linear in $\log_3 j$ (and so is the number of applications of the underlying one-time signature scheme).

In actual applications, one should observe that in variants of Construction 6.4.14 the size of the tree determines the total number of documents that can be signed, whereas in variants of Construction 6.4.16 the tree size has even a more drastic effect on the number of documents that can be signed.[20] In some cases a hybrid of Constructions 6.4.14 and 6.4.16 may be preferable: We refer to a memory-dependent scheme in which leaves are assigned as in Construction 6.4.14 (i.e., according to a counter), but the rest of the operation is done as in Construction 6.4.16 (i.e., the one-time instances are re-generated on-the-fly, rather than being generated and recorded). In some applications, the introduction of a document-counter may be tolerated, and the gain is the ability to use a smaller tree of size merely greater than the total number of documents that should be ever signed.

6.4.3 * Universal One-Way Hash Functions and using them

So far, we have established that *the existence of collision-free hashing collections implies the existence of secure signature schemes* (cf. Corollary 6.4.10). We seek to weaken the assumption under which secure signature schemes can be constructed, and bear in mind that the existence of one-way functions is certainly a necessary condition (cf., for example, Exercise 6). In view of Theorem 6.4.9, we may focus on constructing secure *one-time* signature schemes. Furthermore, recall that secure length-restricted one-time signature schemes can be constructed based on any one-way function (cf. Corollary 6.4.6). Thus, the only bottleneck we face (with respect to the assumption used) is Proposition 6.4.7, which refers to Construction 6.2.6 and utilizes collision-free hashing. Our aim in this section, is to replace this component in the construction. We use a variant of Construction 6.2.6 in which, instead of using collision-free hashing, we use a seemingly weaker notion called *Universal One-Way Hash Functions.*

6.4.3.1 Definition

A collection of universal one-way hash functions is defined analogously to a collection of collision-free hash functions. The only difference is that the hardness (to form collisions) requirement is relaxed. Recall that for a collection of collision-free hash functions it was required that given the function's description it is hard to form an *arbitrary* collision under the function. For a collection of universal one-way hash functions we only require that given the function's description h and a preimage x it is hard to find an $x' \neq x$ so that $h(x') = h(x)$. We refer to this requirement as to *hardness to form designated collisions.*

Our formulation of the hardness to form designated collisions is actually seemingly stronger. Rather than being supplied with a (random) preimage x,

[20] In particular, the number of documents that can be signed should definitely be smaller than the square root of the size of the tree (or else two documents are likely to be assigned the same leaf). Furthermore, we cannot use a small tree (e.g., of size 1000) even if we know that the total number of documents that will ever be signed is small (e.g., 10), since otherwise the probability that two documents are assigned the same leaf is too big.

the collision-forming algorithm is allowed to select x by itself, but must do so before being presented with the function's description. That it, the attack of the collision-forming algorithm proceeds in three stages: first the algorithm selects a preimage x, next it is given a description of a randomly selected function h, and finally it is required to output $x' \neq x$ such that $h(x') = h(x)$. We stress that the third stage in the attack is also given the random choices made while producing the preimage in the first stage. This yields the following definition, where the first stage is captured by a deterministic polynomial-time algorithm A_0 (which maps a sequence of coin tosses, denoted $U_{q(n)}$, to a preimage) and the third stage is captured by algorithm A (which is given the very same $U_{q(n)}$ as well as the function's description).

Definition 6.4.18 (universal one-way hash functions – UOWHF): *Let $\ell : \mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^* \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is called* universal one-way hashing (UOWHF) *if there exists a probabilistic polynomial-time algorithm I so that the following holds*

1. (admissible indexing – technical):[21] *For some polynomial p, all sufficiently large n's and every s in the range of $I(1^n)$ it holds that $n \leq p(|s|)$.*
2. (efficient evaluation): *There exists a polynomial-time algorithm that given s and x, returns $h_s(x)$.*
3. (hard to form designated collisions): *For every polynomial q, every deterministic polynomial-time algorithm A_0, every probabilistic polynomial-time algorithm A, every polynomial p and all sufficiently large n's*
$$\mathsf{Pr}\left[\begin{array}{c} h_{I(1^n)}(A(I(1^n),U_{q(n)})) = h_{I(1^n)}(A_0(U_{q(n)})) \\ \text{and} \quad A(I(1^n),U_{q(n)}) \neq A_0(U_{q(n)}) \end{array}\right] < \frac{1}{p(n)} \qquad (6.5)$$
where the probability is taken over $U_{q(n)}$ and the internal coin tosses of algorithms I and A.

The function ℓ is called the range specifier *of the collection.*

We stress that the *hardness to form designated collisions condition* refers to the following three stage process: first, using a uniformly distributed $r \in \{0,1\}^{q(n)}$, the adversary generates a preimage $x = A_0(r)$; next, a function h is selected; and, finally, the adversary A is given h (as well as r used in the first stage), and tries to find a preimage $x' \neq x$ such that $h(x') = h(x)$. Indeed, Eq. (6.5) refers to the probability that $x' \stackrel{\text{def}}{=} A(h,r) \neq x$ and yet $h(x') = h(x)$.

Note that the range specifier must be super-logarithmic (or else, given s and $x \leftarrow U_n$, one is too likely to find an $x' \neq x$ so that $h_s(x) = h_s(x')$, by uniformly selecting x' in $\{0,1\}^n$). Also note that any UOWHF collection yields a collection of one-way functions (see Exercise 8). Finally, note that any collision-free hashing is universally one-way hashing, but the converse is false (see Exercise 9). Furthermore, it is not known whether collision-free hashing can be constructed based on any one-way functions (in contrast to Theorem 6.4.29 below).

[21] This condition is made merely to avoid annoying technicalities. Note that $|s| = \text{poly}(n)$ holds by definition of I.

6.4.3.2 Constructions

We construct UOWHF collections in several steps, starting with related but restricted notions, and relaxing the restrictions gradually (untill we reach unrestricted UOWHF collections). The restriction we refer to is on the length of the arguments to the function. Most importantly, the hardness (to form designated collisions) requirement will refer only to argument of this length. That is, we refer to the following technical definition.

Definition 6.4.19 ((d,r)-UOWHF): *Let $d,r : \mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{r(|s|)}\}_{s\in\{0,1\}^*}$ is called (d,r)-UOWHF if there exists a probabilistic polynomial-time algorithm I so that the following holds*

1. *For all sufficiently large n's and every s in the range of $I(1^n)$ it holds that $|s| = n$.*[22]
2. *There exists a polynomial-time algorithm that given s and $x \in \{0,1\}^{d(|s|)}$, returns $h_s(x)$.*
3. *For every polynomial q, every deterministic polynomial-time algorithm A_0 mapping $q(n)$-bit long strings to $d(|s|)$-bit long strings, every probabilistic polynomial-time algorithm A, every polynomial p and all sufficiently large n's Eq. (6.5) holds.*

Off course, we care only of (d,r)-UOWHF for functions $d,r : \mathbb{N} \to \mathbb{N}$ satisfying $d(n) > r(n)$. (The case $d(n) \le r(n)$ is trivial since collisions can be avoided altogether; say by the identity map.) The "minimal" non-trivial case is when $d(n) = r(n) + 1$. Indeed, this is our starting point. In fact, the current step is the least obvious step to be taken on our way towards the construction of full-fledged UOWHF.

Step I: constructing $(d,d-1)$-UOWHFs. We show how to construct length-restricted UOWHF that shrink their input by a single bit. Our construction can be carried out using any one-way permutation. In addition, we use a family of hashing functions, S_n^{n-1}, as defined in Section 3.5.1.1. Recall that a function selected uniformly in S_n^{n-1} maps $\{0,1\}^n$ to $\{0,1\}^{n-1}$ in a pairwise independent manner, that the functions in S_n^{n-1} are easy to evaluate, and that for some polynomial p it holds that $\log_2 |S_n^{n-1}| = p(n)$.

Construction 6.4.20 (a $(d,d-1)$-UOWHF): *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a 1-1 and length preserving function, and let S_n^{n-1} be a family of hashing functions such that $\log_2 |S_n^{n-1}| = p(n)$, for some polynomial p. (Specifically, suppose that $\log_2 |S_n^{n-1}| \in \{3n-2, 2n\}$, as in Exercises 22.2 and 23 of Chapter 3.) Then, for*

[22] Here we chose to make a more stringent condition, requiring that $|s| = n$ rather than $n \le \mathrm{poly}(|s|)$. In fact, one can easily enforce this more stringent condition by modifying I into I' so that $I'(1^{l(n)}) = I(1^n)$ for a suitable function $l : \mathbb{N} \to \mathbb{N}$ satisfying $l(n) \le \mathrm{poly}(n)$ and $n \le \mathrm{poly}(l(n))$.

every $s \in S_n^{n-1} \equiv \{0,1\}^{p(n)}$ and every $x \in \{0,1\}^n$, we define $h'_s(x) \stackrel{\rm def}{=} h_s(f(x))$. In case $|s| \notin \{p(n) : n \in \mathbb{N}\}$, we define $h'_s(x) \stackrel{\rm def}{=} h'_{s'}$ where s' is the longest prefix of s satisfying $|s'| \in \{p(n) : n \in \mathbb{N}\}$. We refer to an index selection algorithm that, on input 1^m, uniformly selects $s \in \{0,1\}^m$.

That is, $h'_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{d(|s|)-1}$, where $d(m)$ is the largest integer n satisfying $p(n) \le m$.

The analysis presented below uses, in an essential way, an additional property of the above-mentioned families of hashing functions; specifically, we assume that give two preimage–image pairs it is easy to uniformly generate a hashing function (in the family) that is consistent with these two mapping conditions. Furthermore, to facilitate the analysis we use a specific family of hasing functions, presented in Exercise 23 of Chapter 3: functions in S_n^{n-1} are described by a pair of elements of the finite field $\mathrm{GF}(2^n)$ so that the pair (a,b) describes the function $h_{a,b}$ that maps $x \in \mathrm{GF}(2^n)$ to the $(n-1)$-bit prefix of the n-bit representation of $ax+b$, where the arithmetics is of the field $\mathrm{GF}(2^n)$. This specific family satisfies all additional properties required in the next proposition.

Proposition 6.4.21 *Suppose that f is a one-way permutation, and that* $\log_2 |S_n^{n-1}| = 2n$. *Furthermore, suppose that all but a negligible fraction of the functions in S_n^{n-1} are 2-to-1, and that there exists a probabilistic polynomial-time algorithm that given $y_1, y_2 \in \{0,1\}^n$ and $z_1, z_2 \in \{0,1\}^{n-1}$, outputs a uniformly distributed element of $\{s \in S_n^{n-1} : h_s(y_i) = z_i \ \forall i \in \{1,2\}\}$. Then $\{h'_s\}_{s\in\{0,1\}^*}$ as in Construction 6.4.20 is a $(d, d-1)$-UOWHF, for $d(m) = \lfloor m/2 \rfloor$.*

Proof Sketch: Intuitively, forming designated collisions under $h'_s \equiv h_s \circ f$ yields ability to invert f. Specifically, if on input x' and h'_s one can find an $x \neq x'$ so that $h'_s(x) = h'_s(x')$ then one basically inverts f on $y = f(x)$ (by generating x' and s so that $h_s(y) = h_s(f(x'))$, and trying to form a designated collision with the preimage x'). Thus, with a suitable random choice of s (i.e., so that $h_s(f(x')) = h_s(y)$, where x' is selected before s), we can invert f on a random preimage y.

The actual proof is by a reducibility argument. Suppose that we are given a probabilistic polynomial-time algorithm A' that forms designated collisions under $\{h'_s\}$, with respect to preimages produced by a deterministic polynomial-time algorithm A'_0 that maps $p(n)$-bit strings to n-bit strings. Then, we construct an algorithm A that inverts f. On input $y = f(x)$, where $n = |y| = |x|$, algorithm A proceeds as follows.

(1) Select r' uniformly in $\{0,1\}^{p(n)}$, and compute $x' = A'_0(r')$ and $y' = f(x')$.
(2) Select s uniformly in $\{s \in S_n^{n-1} : h_s(y') = h_s(y)\}$.
(Recall that y is the input to A, and y' is generated by A in Step (1).)
(3) Invoke A' on input (s, r'), and output whatever A' does.

By the second extra condition regarding S_n^{n-1}, Step (2) can be implemented in probabilistic polynomial-time.

Turning to the analysis of algorithm A, we consider the behavior of A on input $y = f(x)$ for a uniformly distributed $x \in \{0,1\}^n$ (which implies that y is uniformly distributed over $\{0,1\}^n$). We first observe that for every fixed r' selected in Step (1), if y is uniformly distributed in $\{0,1\}^n$ then s as determined in Step (2) is uniformly distributed in S_n^{n-1}. Using the first extra condition regarding S_n^{n-1}, it follows that the probability that h_s is not 2-to-1 is negligible. By the construction of A, the probability that $f(x') = y$ is also negligible (but we could have taken advantage of this case too, by augmenting Step (1) so that if $y' = y$ then A halts with output x'). We now claim that, in case $f(x') \neq y$ and h_s is 2-to-1, if A' returns x'' so that $x'' \neq x'$ and $h'_s(x'') = h'_s(x')$ then $f(x'') = y$.

> The claim is proven as follows: By definitions of h'_s and A (i.e., its Step (2)), we have $h'_s(x) = h_s(f(x)) = h_s(f(x')) = h'_s(x')$, which equals $h'_s(x'')$ by one of the claim's hypotheses. By other two hypotheses $x' \neq x''$ and h_s is 2-to-1. Thus, $x' \neq x''$ are the only two preimages of $h_s(y) = h'_s(x)$ under h_s, and so $x \in \{x', x''\}$. Using the last of the hypotheses (i.e., $y = f(x) \neq f(x')$) and the fact that f is 1-1, it follows that $x \neq x'$, which in turn implies $x = x''$ and $y = f(x'')$.

We conclude that if A' forms designated collisions with probability $\varepsilon'(n)$ then A inverts f with probability $\varepsilon'(n) - \mu(n)$, where μ is a negligible function. The proposition follows. $\square$

Step II: constructing $(d', d'/2)$-UOWHFs. We now take the second step on our way, and use any $(d, d-1)$-UOWHF in order to constact a $(d', d'/2)$-UOWHF. That is, we construct length-restricted UOWHF that shrink their input by a factor of 2. For simplicity, we assume that the function $d : \mathbb{N} \to \mathbb{N}$ is onto and monotonically non-decreasing. In such a case we denote by $d^{-1}(m)$ the smallest natural number n satisfying $d(n) = m$.

Construction 6.4.22 (a $(d', d'/2)$-UOWHF): *Let $\{h_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{d(|s|)-1}\}_{s \in \{0,1\}^*}$, where $d : \mathbb{N} \to \mathbb{N}$ is onto and non-decreasing. Then, for every $s_1, ..., s_{\lfloor d(n)/2 \rfloor}$, where each $s_i \in \{0,1\}^{d^{-1}(d(n)+1-i)}$, and every $x \in \{0,1\}^{d(n)}$, we define*

$$h'_{s_1,\ldots,s_{\lfloor d(n)/2 \rfloor}}(x) \stackrel{\text{def}}{=} h_{s_{\lfloor d(n)/2 \rfloor}}(\cdots h_{s_1}(x) \cdots)$$

That is, we let $x_0 \stackrel{\text{def}}{=} x$, and $x_i \leftarrow h_{s_i}(x_{i-1})$, for $i = 1, ..., \lfloor d(n)/2 \rfloor$. (Note that $d(|s_i|) = d(n) + 1 - i$ and $|x_i| = d(n) + 1 - i$ indeed hold.) *We refer to an index selection algorithm that, on input 1^m, determines the largest integer n so that $m \geq m' \stackrel{\text{def}}{=} \sum_{i=1}^{\lfloor d(n)/2 \rfloor} d^{-1}(d(n)+1-i)$, uniformly selects $s_1, ..., s_{\lfloor d(n)/2 \rfloor}$ so that $s_i \in \{0,1\}^{d^{-1}(d(n)+1-i)}$, and $s_0 \in \{0,1\}^{m-m'}$, and lets $h'_{s_0,s_1,\ldots,s_{\lfloor d(n)/2 \rfloor}} \stackrel{\text{def}}{=} h'_{s_1,\ldots,s_{\lfloor d(n)/2 \rfloor}}$.*

That is, $m = |\overline{s}|$ and $h'_{\overline{s}} : \{0,1\}^{d(n)} \to \{0,1\}^{\lceil d(n)/2 \rceil}$, where n is largest so that $m \geq \sum_{i=1}^{\lfloor d(n)/2 \rfloor} d^{-1}(d(n)+1-i)$. Thus, $d'(m) = d(n)$, where n is as above; that is, we have $h'_{\overline{s}} : \{0,1\}^{d'(|\overline{s}|)} \to \{0,1\}^{\lfloor d'(|\overline{s}|)/2 \rfloor}$, with $d'(|\overline{s}|) = d(n)$. Note that, for $d(n) = \Omega(n)$ (as in Construction 6.4.20), it holds that $d'(O(n^2)) \geq d(n)$ and $d'(m) = \Omega(\sqrt{m})$ follows. More generally, if for some polynomial p it holds that $p(d(n)) \geq n$ (for all n's) then for some polynomial p' it holds that $p'(d'(m)) \geq m$ (for all m's), since $d'(p(n) \cdot d(n)) \geq d(n)$. We call a function sufficiently-growing; that is, $d : \mathbb{N} \to \mathbb{N}$ is sufficiently-growing if there exists a polynomial p so that for every n it holds that $p(d(n)) \geq n$.

Proposition 6.4.23 *Suppose that $\{h_s\}_{s \in \{0,1\}^*}$ is a $(d, d-1)$-UOWHF, where $d : \mathbb{N} \to \mathbb{N}$ is onto, non-decreasing and sufficiently-growing. Then, for some sufficiently-growing function $d' : \mathbb{N} \to \mathbb{N}$, Construction 6.4.22 is a $(d', \lfloor d'/2 \rfloor)$-UOWHF.*

Proof Sketch: Intuitively, a designated collision under $h'_{s_1,...,s_{d/2}}$ yields a designated collision under one of the h_{s_i}'s. That is, let $x_0 \stackrel{\text{def}}{=} x$, and $x_i \leftarrow h_{s_i}(x_{i-1})$, for $i = 1, ..., \lfloor d(n)/2 \rfloor$. Then if given x and $\overline{s} = (s_1, ..., s_{d/2})$, one can find an $x' \neq x$ so that $h'_{\overline{s}}(x) = h'_{\overline{s}}(x')$, then there exists an i so that $x_{i-1} \neq x'_{i-1}$ and $h_{s_i}(x_{i-1}) = h_{s_i}(x'_{i-1})$, where the x'_i's are defined analogously to the x_i's. Thus, we obtain a designated collision under h_{s_i}.

The actual proof uses the hypothesis that it is hard to form designated collisions when one is also given the coins used in the generation of the preimage (and not merely the preimage itself). Specifically, we construct an algorithm that forms designated collision under one of the h_{s_i}'s, when given not only x_{i-1} but rather also x_0 (which yields x_{i-1} as above). The proof is by a reducibility argument. We are given a probabilistic polynomial-time algorithm A' that forms designated collisions under $\{h'_{\overline{s}}\}$, with respect to preimages produced by a deterministic polynomial-time algorithm A'_0 that maps $p'(n)$-bit strings to n-bit strings. We construct algorithms A_0 and A so that A forms designated collisions under $\{h_s\}$ with respect to preimages produced by algorithm A_0, which maps $p(n)$-bit strings to n-bit strings, for a suitable polynomial p. Specifically, $p : \mathbb{N} \to \mathbb{N}$ is 1-1 and $p(n) \geq p'(d^{-1}(2d(n))) + n + n \cdot d^{-1}(2d(n))$.

We strat with the description of A_0, which at this point may seem strange. On input $r \in \{0,1\}^{p(n)}$, algorithm A_0 proceeds as follows, where $q(n) \stackrel{\text{def}}{=} d^{-1}(2d(n))$.

Write $r = r_1 r_2 r_3$ so that $|r_1| = n$ and $|r_3| = p'(q(n))$.

(1) Using r_1, determine m in $\{n+1, ..., n \cdot q(n)\}$ and $j \in \{1, ..., q(n)\}$ so that both m and j are almost uniformly distributed in the corresponding sets.

(2) Compute the largest integer n' so that $m \leq \sum_{i=1}^{\lfloor d(n')/2 \rfloor} d^{-1}(d(n') + 1 - i)$.

(3) If $d^{-1}(d(n') + 1 - j) \neq n$ then output the $d(n)$-bit long suffix of r_3. (Comment: the output in this case is immaterial to our proof.)

(4) Otherwise (i.e., $n = d^{-1}(d(n') + 1 - j)$, which is the case we care about), do:

(4.1) Let $s_0 s_1 \cdots s_{j-1}$ be a prefix of r_2 so that
$$|s_0| = m - \sum_{i=1}^{\lfloor d(n')/2 \rfloor} d^{-1}(d(n') + 1 - i),$$
and $|s_i| = d^{-1}(d(n') + 1 - i)$, for $i = 1, ..., j-1$.

(4.2) Let $x_0 \leftarrow A'_0(r')$, where r' is the $p'(d^{-1}(d(n')))$-bit long suffix of r_3.

(4.3) For $i = 1, ..., j-1$, compute $x_i \leftarrow h_{s_i}(x_{i-1})$.
Output x_{j-1}.

As stated above, we only care about the case in which Step (4) is applied. This case occurs with noticeable probability, and the description of the following algorithm A refers to it. On input $s \in \{0,1\}^n$ and $r \in \{0,1\}^{p(n)}$, algorithm A proceeds as follows.

(1–2) Using r, determine m, j and n' exactly as done by A_0.
(3) If $d^{-1}(d(n')+1-j) \neq n$ then abort.
(4) Otherwise (i.e., $n = d^{-1}(d(n')+1-j)$), do:
(4.1) Determine $s_0, s_1, ..., s_{j-1}$ and r' exactly as A_0 does in Step (4).
(4.2) Uniformly select $s_{j+1}, ..., s_{\lfloor d(n')/2 \rfloor}$ so that $s_i \in \{0,1\}^{d^{-1}(d(n')+1-i)}$, and set $s' = s_0, s_1, ..., s_{j-1}, s, s_{j+1}, ..., s_{\lfloor d(n')/2 \rfloor}$.
(4.3) Invoke A' on input (s', r'), and output whatever A' does.

Clearly, if algorithms A' and A'_0 run in polynomial-time then so do A and A_0. We now lower bound the probability that A succeeds to form designated collisions under $\{h_s\}$, with respect to images produced by A_0. We start from the contradiction hypothesis by which the corresponding probability for A' (w.r.t A'_0) is non-negligible.

Let use denote by $\varepsilon'(m)$ the success probability of A' on uniformly distributed input $(s', r') \in \{0,1\}^m \times \{0,1\}^{p'(m)}$. Let n' be the largest integer so that $m \leq \sum_{i=1}^{\lfloor d(n')/2 \rfloor} d^{-1}(d(n')+1-i)$. Then, there exists a $j \in \{1, ..., d(n')\}$ so that with probability at least $\varepsilon'(m)/d'(n')$ on input (s', r'), where $s' = s_0, s_1, ..., s_{\lfloor d(n')/2 \rfloor}$ as above, A' outputs an $x' \neq x \stackrel{\text{def}}{=} A'_0(r')$ so that $h_{s_{j-1}}(\cdots(h_{s_1}(x')\cdots) \neq h_{s_{j-1}}(\cdots(h_{s_1}(x')\cdots)$ and $h_{s_j}(\cdots(h_{s_1}(x')\cdots) = h_{s_j}(\cdots(h_{s_1}(x')\cdots)$. Fixing this m, j and n', let $n = d^{-1}(d(n')+1-j)$, consider what happens when A is invoked on uniformly distributed $(s, r) \in \{0,1\}^n \times \{0,1\}^{p(n)}$. With probability at least $1/m^2$ over the possible r's, the values of m and j are determined to equal the above. Conditioned on this case, A' is invoked on uniformly distributed input $(s', r') \in \{0,1\}^m \times \{0,1\}^{p'(m)}$, and so a collision at the j^{th} hashing function occurs with probability at least $\varepsilon'(m)/d'(n')$. Note that $m = \text{poly}(n)$ and $d'(n') = \text{poly}(n)$. This implies that A succeeds with probability at least $\varepsilon(n) \stackrel{\text{def}}{=} \frac{\varepsilon'(\text{poly}(n))}{\text{poly}(n)}$, with respect to images produced by A_0. Thus, if ε' is non-negligible then so is ε, and the proposition follows. $\square$

Step III: Length-restricted UOWHFs that shrink each input by a factor of two. The third step on our way consists of using any $(d, d/2)$-UOWHF in order to construct length-restricted UOWHFs that are applicable to any input length but shrink each input to half it length (rather than to a fixed length that only depends on the function description). The resulted construct does not fit Definition 6.4.19, since it can be applied to any input length (rather than only to a single length determined by the function's description). Yet, the resulting construct yields a $(d', d'/2)$-UOWHF for any polynomially-bounded function d', whereas in Construction 6.4.22 the function d' satisfies $d'(n) \ll n$.

Construction 6.4.24 (a $(d', d'/2)$-UOWHF for any d'): *Let $\{h_s : \{0,1\}^{d(|s|)} \to \{0,1\}^{\lfloor d(|s|)/2 \rfloor}\}_{s \in \{0,1\}^*}$, where $d : \mathbb{N} \to \mathbb{N}$ is onto and non-decreasing. Then, for every $s \in \{0,1\}^n$, every $t \in \mathbb{N}$ and every $x \in \{0,1\}^*$, we define*

$$h'_s(x) \stackrel{\text{def}}{=} h_s(x_1) \cdots h_s(x_t 1 0^{d(n)-|x_t|-1})$$

where $x = x_1 \cdots x_t$, $0 \leq |x_t| < d(n)$ and $|x_i| = d(n)$ for $i = 1, ..., t-1$. The index selection algorithm of $\{h'_s\}$ is identical to the one of $\{h'_s\}$.

Clearly, Construction 6.4.24 satisfies Conditions 1 and 2 of Definition 6.4.18, provided that $\{h_s\}$ satisfies the corresponding conditions of Definition 6.4.19. We thus focus of the hardness to form designated collisions property.

Proposition 6.4.25 *Suppose that $\{h_s\}_{s \in \{0,1\}^*}$ is a $(d, d/2)$-UOWHF, where $d : \mathbb{N} \to \mathbb{N}$ is onto, non-decreasing and sufficiently-growing. Then Construction 6.4.22 satisfies Condition 3 of Definition 6.4.18.*

Proof Sketch: Intuitively, a designated collision under h'_s yields a designated collision under h_s. That is, consider the parsing of each string into blocks of length $d(n)$, as in the above construction. Then if given $x = x_1 \cdots x_t$ and s, one can find an $x' = x'_1 \cdots x'_{t'} \neq x$ so that $h'_s(x) = h'_s(x')$, then $t' = t$ and there exists an i so that $x_i \neq x'_i$ and $h_s(x_i) = h_s(x'_i)$.

The actual proof is by a reducibility argument. Given a probabilistic polynomial-time algorithm A' that forms designated collisions under $\{h'_s\}$, with respect to images produced by a deterministic polynomial-time algorithm A'_0, we construct algorithms A_0 and A so that A forms designated collisions under $\{h_s\}$ with respect to images produced by algorithm A_0. Specifically, algorithm A_0 invokes A'_0, and uses extra randomness (supplied in its input) to uniformly select one of the $d(n)$-bit long blocks in the standard parsing of the output of A'_0. That is, the random-tap used by algorithm A_0 has the form (r', i), and A_0 outputs the i^{th} block in the parsing of $A'_0(r')$. Algorithm A is obtained analogously. That is, given $s \in \{0,1\}^n$ and coins $r = (r', i)$ used by A_0, algorithm A invokes A' on input s and r', obtains the output x', and outputs the i^{th} block in the standard parsing of x'.

Clearly, if algorithm A' succeeds (in forming designated collisions w.r.t $\{h'_s\}$) with probability $\varepsilon'(n)$ then algorithm A succeeds (in forming designated collisions w.r.t $\{h_s\}$) with probability at least $\varepsilon'(n)/t(n)$, where $t(n)$ is a bound on the running-time of A' (which also bounds the length of the output of A'). The proposition follows. $\square$

Step IV: Full-fledged UOWHFs. The last step on our way consists of using any length-restricted UOWHFs as constructed above to obtain full-fledged UOWHFs. That is, we use length-restricted UOWHFs that are applicable to any input length but shrink each input to half its length (rather than to a fixed length that only depends on the function description). The resulted construct is a UOWHF (as defined in Definition 6.4.18).

Construction 6.4.26 (a UOWHF): *Let $\{h_s : \{0,1\}^* \to \{0,1\}^*\}_{s\in\{0,1\}^*}$, so that $|h_s(x)| \leq |x|/2$, for all x's. Then, for every $s_1, ..., s_n \in \{0,1\}^n$, every $t \in \mathbb{N}$ and $x \in \{0,1\}^{2^t \cdot n}$, we define*

$$h'_{s_1,\ldots,s_n}(x) \stackrel{\rm def}{=} h_{s_t}(\cdots h_{s_1}(x)\cdots)$$

That is, we let $x_0 \stackrel{\rm def}{=} x$, and $x_i \leftarrow h_{s_i}(x_{i-1})$, for $i = 1, ..., t$. Strings x of length that is not of the form $2^t \cdot n$ are padded into such strings in a standard manner. We refer to an index selection algorithm that, on input 1^m, determines $n = \lfloor\sqrt{m}\rfloor$, uniformly selects $s_1, ..., s_n \in \{0,1\}^n$ and $s_0 \in \{0,1\}^{m-n^2}$, and lets $h'_{s_0,s_1,\ldots,s_n} \stackrel{\rm def}{=} h'_{s_1,\ldots,s_n}$.

Note that $h'_{s_1,\ldots,s_n} : \{0,1\}^* \to \{0,1\}^n$.

Proposition 6.4.27 *Suppose that $\{h_s\}_{s\in\{0,1\}^*}$ satisfies the conditions of Definition 6.4.18, except that it maps arbitrary input strings to outputs having half the length (rather than a length determined by $|s|$). Then Construction 6.4.26 constritues a collection of UOWHF.*

Proof Sketch: The proof is almost identical to the proof of Proposition 6.4.23. □

Conclusion: Combining Proposition 6.4.21, 6.4.23, 6.4.25 and 6.4.27, we obtain:

Theorem 6.4.28 *If one-way permutations exist then universal one-way hash functions exist.*

Note that the *only* barrier towards constructing UOWHF based on *arbitrary* one-way functions is Proposition 6.4.21, which refers to one-way *permutations*. In fact, if we wish to construct UOWHF based on any one-way function then we need to present an alternative construction of $(d, d-1)$-UOWHF (i.e., an alternative to Construction 6.4.20 which fails in case f is 2-to-1).[23] Such a construction is actually known, and so the following result is known to hold (but is not proven here):

Theorem 6.4.29 *Universal one-way hash functions exist if and only if one-way functions exist.*

We stress that the difficult direction is the one referred to above (i.e., from one-way functions to UOWHF collections). For the much easier converse, see Exercise 8.

[23] For example, if $f(\sigma, x') = (0, f'(x'))$, for $\sigma \in \{0,1\}$, then forming designated collisions under Construction 6.4.20 is easy: Given $(0, x')$, one outputs $(1, x')$, and indeed a collision is formed already under f.

6.4.3.3 One-time signature schemes based on UOWHF

Using universal one-way hash functions, we present an alternative construction of one-time signature schemes based on length-restricted one-time signature schemes. Specifically, we replace Construction 6.2.6 in which collision-free hashings were used by the following construction in which universal one-way hash functions are used instead. The difference between the two constructions is that here the (description of the) hashing function is not a part of the signing and verification keys, but is rather selected on the fly by the signing algorithm (and appears as part of the signature). Furthermore, the description of the hash function is being authenticated (by the signer) together with the hash function. It follows that the forging adversary, which is unable to break the length-restricted one-time signature scheme, must form a designated collision (rather than an arbitrary one). However, the latter is infeasible too (by virtue of the UOWHF collection in use). We comment that the same (new) construction is applicable to length-restricted signature schemes (rather than to one-time ones): we stress that, in this case, a new hashing function is selected at random each time the signing algorithm is applied. In fact, we present the more general construction.

Construction 6.4.30 (the hash and sign paradigm, revisited): *Let $\ell : \mathbb{N} \to \mathbb{N}$ and (G, S, V) be an ℓ-restricted signature scheme as in Definition 6.2.1, and let $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)-|r|}\}_{r\in\{0,1\}^*}$ be a UOWHF as in Definition 6.4.18. We construct a general signature scheme, (G', S', V'), with G' identical to G, as follows:*

signing with S': *On input a signing-key $s \in G'_1(1^n)$ and a document $\alpha \in \{0,1\}^*$, algorithm S' proceeds in two steps:*

1. *Algorithm S' invokes I, the indexing algorithm of the UOWHF collection, to obtain $\beta' \leftarrow I(1^n)$.*
2. *Algorithm S' invokes S* (once) *to produce $\beta'' \leftarrow S_s(\beta', h_{\beta'}(\alpha))$.*

Algorithm S' outputs (β', β'').

verification with V': *On input a verifying-key $v \in G'_2(1^n)$, a document $\alpha \in \{0,1\}^*$, and a alledged signature (β', β''), algorithm V' invokes V, and outputs $V_v((\beta', h_{\beta'}(\alpha)), \beta'')$.*

Recall that secure ℓ-restricted one-time signature schemes exist for any polynomial ℓ, provided that one-way function exist. Thus, the fact that the above construction requires $\ell(n) \gg n$ is not a problem.

Proposition 6.4.31 *Suppose that (G, S, V) is a secure ℓ-restricted signature scheme and that $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)-|r|}\}_{r\in\{0,1\}^*}$ is indeed a collection of UOWHF. Then (G', S', V'), as defined in Construction 6.2.6, is a secure* (full-fledged) *signature scheme. Furthermore, if (G, S, V) is only a secure ℓ-restricted one-time signature scheme then (G', S', V') is a secure one-time signature scheme.*

Proof Sketch: The proof follows the underlying principles of the proof of Proposition 6.2.7. That is, forgery with respect to (G', S', V') yields either forgey with respect to (G, S, V) or a collision under the hash function, where in the latter case a designated collision is formed (in contradiction to the hypothesis regarding the UOWHF). For the furthermore-part, the observation underlying the proof of Proposition 6.4.7 still holds. Details follow.

Given an adversary A' attacking the complex scheme (G', S', V'), we construct an adversary A that attacks the ℓ-restricted scheme, (G, S, V). The adversary A uses I (the indexing algorithm of the UOWHF collection) and its oracle S_s in order to emulate the oracle S'_s for A'. This is done in a straightforward manner; that is, algorithm A emulates S'_s by using the oracle S_s (exactly as S'_s actually does). That is, to answer query α, algorithm A generates $\beta' \leftarrow I(1^n)$ forwards $(\beta', h_{\beta'}(\alpha))$ to its own oracle (i.e., S_s), and answers with (β', β''), where $\beta'' \leftarrow S_s(\beta', h_{\beta'}(\alpha))$. When A' outputs a document-signature pair relative to the complex scheme (G', S', V'), algorithm A tries to use it in order to form a document-signature pair relative to the ℓ-restricted scheme, (G, S, V).

Assume that with (non-negligible) probability $\varepsilon'(n)$, the (probabilistic polynomial-time) algorithm A' succeeds in existentially forging relative to the complex scheme (G', S', V'). Let (α_i, β_i) denote the i^{th} query and answer pair made by A', and (α, β) be the forged document-signature pair that A' outputs (in case of success), where $\beta_i = (\beta'_i, \beta''_i)$ and $\beta = (\beta', \beta'')$. We consider the following cases regarding the forging event:

1. $(\beta', h_{\beta'}(\alpha)) \neq (\beta'_i, h_{\beta'_i}(\alpha_i))$ for all i's. (That is, the S_s-signed value in the forged signature is different from all values used in the queries to S_s.) In this case, the pair $((\beta', h_{\beta'}(\alpha)), \beta'')$ constitutes a success in existential forgery relative to the ℓ-restricted scheme.

2. $(\beta', h_{\beta'}(\alpha)) = (\beta'_i, h_{\beta'_i}(\alpha_i))$ for some i. (That is, the S_s-signed value used in the forged signature equals the i^{th} query made to S_s, although $\alpha \neq \alpha_i$.) Thus, $\beta' = \beta'_i$ and $h_{\beta'}(\alpha) = h_{\beta'_i}(\alpha_i)$, although $\alpha \neq \alpha_i$. In this case, the pair (α, α_i) forms a designated collision under $h_{\beta'_i}$ (and we do not obtain success in existential forgery relative to the ℓ-restricted scheme).

Thus, if the first case occurs with probability at least $\varepsilon'(n)/2$ then A succeeds in its attack on (G, S, V) with probability at least $\varepsilon'(n)/2$, which contradicts the security of the ℓ-restricted scheme (G, S, V). On the other hand, if the second case occurs with probability at least $\varepsilon'(n)/2$ then we derive a contradiction to the difficulty of forming designated collision with respect to $\{h_r\}$. Details (regarding the second case) follow.

We start with a sketch of a construction of an algorithm that attempts to form designated collisions under a uniformly selected hash function. Recall that such an algorithm operates in three stages (see discussion preceding Definition 6.5): first the algorithm selects a preimage x, next it is given a description of a function h, and finally it is required to output $x' \neq x$ such that $h(x') = h(x)$. We stress that the third stage in the attack is also given the random choices made while producing the preimage x in the first stage. Loosely speaking, we construct an algorithm B' that tries to form designated collisions by selecting at random the index i of the query of A' for which the S_s-signed value used in the forged signature equals the i^{th} query made to S_s. Algorithm B' will generate $(s,v) \leftarrow G'(1^n)$, and emulate the attack of A' on S'_s, while also answering the queries of S'_s. In particular, all queries except the i^{th} one are emulated in the straightfoward manner (i.e., by executing the program of S'_s as stated). The i^{th} query of A' will be used as the designated preimage: once it is issued, B' completes its first stage, and obtains a description of a random hashing function h_r (thus completing its second operation stage). Next, algorithm B' answers the i^{th} query, denoted α_i, by applying S_s to $(r, h_r(\alpha_i))$. (This operation belongs to the third stage of B'.) As siad above, subsequent queries are emulated in the straightfoward manner (but this done by the third stage of B', in contrast to the $i-1$ first queries that are handled by the first stage of B'). When A' halts, B' checks whether A' has output a valid signature as in the second case above, and whether the collision formed is indeed on the i^{th} query. When this happens, B' has succeeded in forming a designated collision. In particular, if the second case occurs with probability at least $\frac{\varepsilon'(n)}{2}$ and A' makes at most $t(n)$ queries then B' succeeded in forming a designated collision with probability at least $\frac{1}{t(n)} \cdot \frac{\varepsilon'(n)}{2}$, which contradicts the hypothesis that $\{h_r\}$ is UOWHF.

The furthermore part of the proposition follows by observing that if the forging algorithm A' makes at most one query then the same holds for the algorithm A constructed above. Thus, if (G', S', V') can be broken via a single-message attack that either (G, S, V) can be broken via a single-message attack or one can form designated collisions (w.r.t $\{h_r\}$). In both cases, we reach a contradiction. □

Author's Note: *Should I augment the above proof sketch?*

Conclusion: Combining the furthermore-part of Proposition 6.4.31, Corollary 6.4.6, and the fact that UOWHF collections imply one-way functions (see Exercise 8), we obtain:

Theorem 6.4.32 *If there exist universal one-way hash functions then secure one-time signature schemes exist too.*

6.4.3.4 Conclusions and comments

Combining Theorems 6.4.28, 6.4.32 and 6.4.9, we obtain:

Corollary 6.4.33 *If one-way permutations exists then there exist secure signature schemes.*

Like Corollary 6.4.10, Corollary 6.4.33 asserts the existence of secure (public-key) signature schemes, based on an assumption that does *not* mention trapdoors. Furthermore, the assumption made in Corollary 6.4.33 seems weaker than the one made in Corollary 6.4.10. We can further weaker the assumption by using Theorem 6.4.29 (which was stated without a proof) rather than Theorem 6.4.28. Specifically, combining Theorems 6.4.29, 6.4.32 and 6.4.9, we establish Theorem 6.4.1. That is, *secure signature schemes exist if and only if one-way functions exist*.

> Author's Note: *Further discuss the revised hash-then-sign paradigm. That is, call attention to Construction 6.4.30...*

6.5 Miscellaneous

6.5.1 Historical Notes

The notion of a (public-key) signature scheme was introduced by Diffie and Hellman [62], who also suggested to implement it using trapdoor permutations. Concrete implementations were suggested by Rivest, Shamir and Adleman [191] and by Rabin [187]. However, definitions of security for signature schemes were presented only a few years afterwards.

A first rigorous treatment of security notions for signature schemes was suggested by Goldwasser, Micali and Yao [127], but their definition is weaker than the one followed in our text (above). (Specifically, the adversary's queries in the definition of [127] are determined non-adaptively and obliviously of the public-key.) Assuming the intractability of factoring, they also presented a signature scheme that is secure under their definition. We stress that the security definition of [127] is natural and significatly stronger than all security notions considered before [127].

A comprehensive treatment of security notions for signature schemes, which culminates in the notion used in our text, was presented by Goldwasser, Micali and Rivest [125]. Assuming the intractability of factoring, they also presented a signature scheme that is secure (in the sense of Definition 6.1.2). This was the first time that a signature scheme was proven secure under a simple intractability assumption such as the intractability of factoring. Their proof has refuted a folklore (attributed to Ron Rivest) by which no such "constructive proof" may exist (as its mere existence was believed to yield a forging procedure). Whereas the (two) schemes of [127] were inherently memory-dependent, the scheme of [125] have a "memoryless" variant (cf. [89] and [125]).

Following [125], research has focused on constructing secure signature schemes under weaker assumptions. In fact, as stated in [125], their construction can be carried out using any collection of claw-free trapdoor permutation pairs. The claw-free requirement was omitted in [20], whereas the trapdoor requirement was omitted by Naor and Yung [175]. Finally, Rompel showed that one may use arbitrary one-way functions rather one-way permutations [192], and thus established Theorem 6.4.1. The progress briefly summarized above was enables by the use of many important ideas and/or paradigms, some of them were introduced in this body of work and some were "only" revisited and properly formalized.

We refer specifically to the introduction of the refreshing paradigm in [125], to the use of authentication trees (cf., [160, 161] and [125]), to the introduction of Universal One-Way Hash Functions in [175], and to the use of one-time signature schemes (cf., [186]).

We comment that our presentation is different from the one available in any of the above cited papers. Specifically, the main part of Section 6.4 is based on a variant of the signature scheme of [175], using collision-free hashing (cf. [57]) instead of universal one-way hashing (cf. [175]).

Message authentication schemes. As in case of encryption schemes, the rigorous study of the security of private-key signature schemes (i.e., message authentication schemes) has legged behind the corresponding study of public-key signature schemes. The construction of message authentication schemes based on pseudorandom functions is due to [103]. (Alternative constructions are presented and discussed in [**?**, **?**, 14, 15].)

6.5.2 Suggestion for Further Reading

As mentioned above, the work of Goldwasser, Micali and Rivest contains a comprehensive treatment of security notions for signature schemes [125]. Their treatment refers to two parameters: (1) the type of attack, and (2) the type of forgery that follows from it. The most severe type of attack allows the adversary to adaptively select the documents to be signed (as in Definition 6.1.2). The most liberal notion of forgey refers to producing a signature to any document for which a signature was not obtained during the attack (again, as in Definition 6.1.2). Thus, the notion of security presented in Definition 6.1.2 is the strongest among the notions discussed in [125]. (Still, in some applications, weaker notions of security may suffice.) We stress that one may benefit from the definitional part of [125], but the constructive part of [125] is should be ignored since it is superseeded by later work (on which our presentation is based).

Pfitzmann's book [183] contains a comprehensive discussion of many aspects involved in the integration of signature schemes in real-life systems. In addition, her book surveys variants and augmentations of the notion of signature schemes, viewing the one treated in the current book as "ordinary". The focus is on "fail-stop" signature schemes [183, Chap. 7–11], but much attention is given to the presentation of a general framework [183, Chap. 5] and to review of other "non-ordinary" schemes [183, Sec. 2.7 & 6.1].

> Author's Note: *For further discussion of message authentication schemes, the reader in referred to [15].*

> Author's Note: *The constructions of universal one-way hash functions presented above use any one-way permutation, in a generic way, so that the number of applications of the one-way permutation is linearly related to the difference between the number of input and output bits in the hash function. In [88], it is shown that as as far*

as generic (black-box) constructions go, this is essentially the best performance that one can hope for.

Author's Note: *Add comment about the scheme of Dwork–Naor (cf. [65]).*

Author's Note: *Add comment about the offline/online signature scheme (cf. [71]).*

6.5.3 Open Problems

The known construction of signature schemes from *arbitrary* one-way functions has no practical significance [192]. It is indeed an important open problem to provide an alternative construction that may be practical and still utilize an *arbitrary* one-way function. We believe that providing such a construction may require the discovery of new important paradigms.

6.5.4 Exercises

Exercise 1: *On the triviality of ℓ-restricted signature schemes, for logarithmically bounded ℓ:*

1. Show that for logarithmically bounded ℓ, secure ℓ-restricted private-key signature schemes (i.e., message authentication schemes) can be constructed, without relying on any assumptions.

   Guideline: On input 1^n, the key generator uniformly selects $s \in \{0,1\}^{2^{\ell(n)}\cdot n}$, and outputs the key pair (s,s). View $s = s_1 \cdots s_{2^{\ell(n)}}$, where each s_i is an n-bit long string, and consider any fixed ordering of the $2^{\ell(n)}$ strings of length $\ell(n)$. The signature to $\alpha \in \{0,1\}^{\ell(n)}$ is defined as s_i, where i is the index of α in the latter ordering.

2. In contrast, show that the existence of a secure ℓ-restricted public-key signature scheme, even for $\ell \equiv 1$, implies the existence of one-way functions.

   Guideline: Let (G,S,V) be a 1-restricted public-key signature scheme. Define $f(1^n,r) = v$ if on input 1^n and coins r, algorithm G generates the key-pair of the form $(\cdot,v)$. Assuming that algorithm A inverts f with probability $\varepsilon(n)$, we construct a forger that attacks (G,S,V) as follows. On input a verification key v, the forger invokes A on input v. With probability $\varepsilon(n)$, the forger obtains r so that $f(1^n,r) = v$. In such a case, the forger obtains a matching signing-key s (i.e., (s,v) is output by $G(1^n)$ on coins r), and so can produce valid signatures (e.g., $S_s(0)$ is accepted by V_v as a signature to 0).

Exercise 2: Using a pseudorandom function ensemble of the form $\{f_s : \{0,1\}^* \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, construct a general secure message authentication scheme (rather than a length-restricted one).

Guideline: The construction is identical to Construction 6.3.1, except that here we use a general pseudorandom function ensemble rather than the one used there. The proof of security is analogous to the proof of Proposition 6.3.2.

Exercise 3: Prove that the existence of secure message authentication schemes implies the existence of one-way functions.

Guideline: First show how to use any message authentication scheme in order to construct a boolean pseudorandom function (as defined in Definition 3.6.12 for the case $r(n) = 1$), and then show that the latter gives rise to a pseudorandom generator (analogously to Exercise 28 of Chapter 3). The first step is the more challenging one: define the function $f_{s,r}$ so that $f_{s,r}(\alpha)$ equals the inner-product mod 2 of r and $S_s(\alpha)$, where S_s is the effect of the signing algorithm with signing-key s. The argument is analogous the proof of Theorem 2.5.2, but is more subtle here (see [174]).

Exercise 4: Prove that, without loss of generality, one can always assume that a chosen message attack makes at least one query. (This holds for general signature schemes as well as for length-restricted and/or one-time ones.)

Guideline: Given an adversary A' that outputs a message-signature pair (α, β) without making any query, modify it so that it makes an arbitrary query $\alpha' \in \{0,1\}^{|\alpha|} \setminus \{\alpha\}$ just before making that output.

Exercise 5: *On the triviality of length-restricted one-time message authentication schemes*: Define one-time and length-restricted one-time message authentication schemes. Show that for any polynomially-bounded and polynomial-time computable function $\ell : \mathbb{N} \to \mathbb{N}$, secure ℓ-restricted one-time message authentication schemes can be constructed, without relying on any assumptions.

Guideline: Combine the ideas underlying Exercise 1 and Construction 6.4.4.

Exercise 6: Prove that the existence of secure one-time signature schemes implies the existence of one-way functions.

Guideline: See guideline for Item 2 in Exercise 1.

Exercise 7: Prove that the existence of collision-free hashing collections implies the existence of one-way functions.

Guideline: Given a collision-free hashing collection, $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell(|r|)}\}_{r \in \{0,1\}^*}$, consider the function $f(r,x) = (r, h_r(x))$, where (say) $|x| = \ell(|r|) + |r|$. Prove that f is a one-way function, by assuming towards the contradiction that f can be efficiently inverted with non-negligible probability, and deriving an efficient algorithm that forms collisions on random h_r's. Given r, form a collision under the function h_r, by uniformly selecting $x \in \{0,1\}^{\ell(|r|)+|r|}$, and feeding the inverting algorithm with input $(r, h_r(x))$. Observe that with non-negligible probability a preimage is obtained, and that with exponentially vanishing probability this preimage is (r,x) itself. Thus, with non-negligible probability, we obtain a preimage $(r,x') \neq (r,x)$ and it holds that $h_r(x') = h_r(x)$.

Exercise 8: Prove that the existence of collections of UOWHF implies the existence of one-way functions.

> **Guideline:** Note that the guidelines provided in Exercise 7 apply here too.

Exercise 9: Assuming the existence of one-way functions, show that there exists a collection of universal one-way hashing functions that is not collision-free.

> **Guideline:** Given a collection of universal one-way hashing functions, $\{f_s : \{0,1\}^* \to \{0,1\}^{|s|}\}$, consider the collection $F' = \{f'_s : \{0,1\}^* \to \{0,1\}^{|s|}\}$ defined so that $f'_s(x) = f_s(x)$ if the $|s|$-bit long prefix of x is different from s, and $f'_s(sx') = s$ otherwise. Clearly, F' is not collision-free. Show that F' remains universal one-way hashing.

Author's Note: *First draft written mainly in May 2000.*