Fragments of a chapter on Signature Schemes

(revised, second posted version)

Extracts from a working draft
for Volume 2 of Foundations of Cryptography

Oded Goldreich

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

February 10, 2002

to Dana

II

Preface

The current manuscript is a preliminary draft of the chapter on signature schemes (Chapter 6) of the second volume of the work *Foundations of Cryptography*. This manuscript subsumes a previous version posted in May 2000.

The bigger picture. The current manuscript is part of a working draft of Part 2 of the three-part work *Foundations of Cryptography* (see Figure 0.1). The three parts of this work are *Basic Tools*, *Basic Applications*, and *Beyond the Basics*. The first part (containing Chapters 1–4) has been published by Cambridge University Press (in June 2001). The second part, consists of Chapters 5–7 (regarding Encryptioni Schemes, Signatures Schemes, and General Cryptographic Protocols, respectively). We hope to publish the second part with Cambridge University Press within a couple of years.

Part 1: Introduction and Basic Tools
  Chapter 1: Introduction
  Chapter 2: Computational Difficulty (One-Way Functions)
  Chapter 3: Pseudorandom Generators
  Chapter 4: Zero-Knowledge Proofs
Part 2: Basic Applications
  Chapter 5: Encryption Schemes
  Chapter 6: Signature Schemes
  Chapter 7: General Cryptographic Protocols
Part 3: Beyond the Basics
  . . .

Figure 0.1: Organization of this work

The partition of the work into three parts is a logical one. Furthermore, it offers the advantage of publishing the first part without waiting for the completion of the other parts. Similarly, we hope to complete the second part within a couple of years, and publish it without waiting for the third part.

Prerequisites. The most relevant background for this text is provided by basic knowledge of algorithms (including randomized ones), computability and elementary probability theory. Background on (computational) number theory, which is required for specific implementations of certain constructs, is not really required here.

Using this text. The text is intended as part of a work that is aimed to serve both as a textbook and a reference text. That is, it is aimed at serving both the beginner and the expert. In order to achieve this aim, the presentation of the basic material is very detailed so to allow a typical CS-undergraduate to follow it. An advanced student (and certainly an expert) will find the pace (in these parts) way too slow. However, an attempt was made to allow the latter reader to easily skip details obvious to him/her. In particular, proofs are typically presented in a modular way. We start with a high-level sketch of the main ideas, and only later pass to the technical details. Passage from high-level descriptions to lower level details is typically marked by phrases such as *details follow*.

> In a few places, we provide straightforward but tedious details in indented paragraphs as this one. In some other (even fewer) places such paragraphs provide technical proofs of claims that are of marginal relevance to the topic of the book.

More advanced material is typically presented at a faster pace and with less details. Thus, we hope that the attempt to satisfy a wide range of readers will not harm any of them.

Teaching. The material presented in the full (three-volume) work is, on one hand, way beyond what one may want to cover in a course, and on the other hand falls very short of what one may want to know about Cryptography in general. To assist these conflicting needs we make a distinction between *basic* and *advanced* material, and provide suggestions for further reading (in the last section of each chapter). In particular, sections, subsections, and subsubsections marked by an asterisk (*) are intended for advanced reading.

Table of 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. Towards presenting the latter constructions, we discuss restricted types of message authentication and signature schemes, which are of independent interest, such as length-restricted schemes (see Section 6.2) and one-time signature schemes (see Section 6.4.1).

Teaching Tip: Indeed, do not skip Section 6.2, since it does play an important role in the following sections. As in Chapter 5, we assume that the reader is familiar with the material in Chapters 2 and 3 (and specifically with Sections 2.2, 2.4, and 3.6). This familiarity is important not only because we use some of the notions and results presented in these sections, but rather because we use similar proof techniques (and do it while assuming that this is *not* the reader's first encounter with these techniques).

6.1 Definitional Issues

Loosely speaking, message authentication and signature schemes are supposed to enable reliable transmission of data between parties. That is, the basic setting consists of a *sender* and a *receiver*, where the receiver may be either predetermined or determined only after the data was sent. Loosely speaking, the receiver wishes to be guaranteed that the data received was actually sent by the sender, rather than modified (or even concocted) by a third party. The receiver may be a party sharing an unreliable point-to-point communication line with the sender (which is indeed the typical setting in which message authentication is employed). However, in other cases (i.e., when signature schemes are employed), the receiver may be any party that obtains the data in the future and wishes to verify that it was indeed sent by the declared sender. In both cases, the reliability (or authenticity) of the data is established by an authentication process that consists of two main processes:

1. A signing process that is employed by the alleged sender in order to produce signatures to data of its choice.
2. A verification process that is employed by the receiver in order to determine the authenticity of the data using the provided signature.

As in case of encryption schemes, the authentication process presupposes also a third (implicit) process called key-generation that allows the sender to generate a signing-key (to be used in the signing process), along with a verification-key (to be used in the verification process). The possession of the signing-key constitutes the sender's advantage over the adversary (see analogous discussion in Chapter 5).

6.1.1 Message authentication versus signature schemes

The difference between message authentication and signature schemes arises from the difference in the settings to which they are intended, which amounts to a difference in the identity of the receiver and in the level of trust that the sender has in the receiver. Typically, message authentication schemes are employed in cases where the receiver is predetermined (at the time of message transmission) and is fully trusted by the sender, whereas signature schemes allow verification of the authenticity of the data by anybody (which is certainly not trusted by the sender). 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. Thus, in signature schemes the verification-key must be known to anybody, and in particular is known to the adversary. In contrast, 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.

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 that lacks a good name (since both authentication and signatures are already taken by one of

type	verification-key known	verification possible
Message auth. schemes	to designated (trusted) receiver(s) only	for designated (trusted) receiver(s) only
Signature schemes	to everybody (including adversary)	for anybody (including adversary)

Figure 6.1: Message authentication versus signature schemes.

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 will be 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 V* (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 $G(1^n)$ constitutes 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 (i.e., insecure) algorithms may satisfy it (e.g., $S(s,\alpha) \stackrel{\rm def}{=} 0$ and $V(v,\alpha,\beta) \stackrel{\rm def}{=} 1$, for all s, v, α and β). 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), whereas 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 (slightly informal) formulation.

- 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 modeled by a probabilistic oracle machine that is given oracle access to a "keyed signing process" (i.e., the signing algorithm combined with the signing-key). Depending on the version (i.e., public-key or not), the attacker may get the corresponding verification-key as input. We stress that this is the *only* difference between the two cases (i.e., private-key and public-key) that are spelled out in Definition 6.1.2. We refer the reader to the clarifying discussion that follows Definition 6.1.2; in fact, some readers may prefer that discussion over the technical formulations.

Definition 6.1.2 (unforgeable signatures):

Common notation: *Let M be a probabilistic oracle 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*

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

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.

[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.

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*

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

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.[2] Next, an attacker is invoked on input 1^n or v, depending if we are in the private-key or public-key case. In both cases, the 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). The attacker 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 $V_v(\alpha, \beta) = 1$. The same applies also in case V is probabilistic, and when viewing $V_v(\alpha, \beta) = 1$ as a random variable. (Alternatively, in the latter case, a condition such as $\mathsf{Pr}[V_v(\alpha, \beta) = 1] \geq 1/2$ may replace the condition $V_v(\alpha, \beta) = 1$.)

6.1.4 Comments

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 (as well as any other "natural" message authentication scheme). Following are a few other comments regarding the definitions.

[2] We stress that $G_1(1^n)$ and $G_2(1^n)$ represent related random variables. Thus, given oracle access to $S_{G_1(1^n)}$ means given oracle access to G_s, where s is selected and fixed according to $G_1(1^n)$.

6.1.4.1 Augmenting the attack with a verification oracle

Indeed, it is natural to augment Definition 6.1.2 by providing the adversary with unlimited access to the corresponding verification oracle V_v. We stress that (in this augmented definition) the documents that (only) appear in the verification queries are not added to the set $Q_M^{S_s}$; that is, the output (α, β) is considered a successful forgery even if the adversary made the verification-query (α, β), but provided (as before) that the adversary did not make the signing-query α (and that $V_v(\alpha, \beta) = 1$).

Indeed, in the public-key case, the verification-oracle adds no power to the adversary, since the adversary (which is given the verification-key) can emulate the verification-oracle by itself. Furthermore, *typically*, also in the private-key model, the verification-oracle does not add much power. Specifically, as discussed in Section 6.5.1 (see also Exercises 1 and 2), any secure private-key signature scheme can be transformed into one having a deterministic verification algorithm and unique valid signatures (i.e., for every verification-key v and document α, there exists a unique β such that $V_s(\alpha, \beta) = 1$). In fact, all private-key signature schemes presented in Section 6.3 have unique valid signatures. Considering an arbitrary combined attack on such a private-key signature scheme, we emulate the verification-queries (in the original model) as follows.

- For a verification-query (α, β) if α equals a previous signing-query, then we can emulate the answer by ourselves. Specifically, if the signing-query α was answered with β then we we answer the verification-query positively else we answer it negatively.

- Otherwise (i.e., for a verification-query (α, β) such that α does not equal any previous signing-query), we may choose either to output (α, β) as a candidate forgery (gambling on $V_v(\alpha, \beta) = 1$) or emulate a negative answer by ourselves (gambling on $V_v(\alpha, \beta) = 0$). Specifically, for every such verification-query, we may choose the first possibility with probability $1/t(n)$ and the second possibility otherwise, where $t(n)$ is a bound on the number of verification-queries performed by the original augmented attack (which we emulate).

For further discussion see Exercise 3.

6.1.4.2 Inessential generalities

The definitions presented above (specifically, Definition 6.1.1) were aimed at generality and flexibility. We comment that several levels of freedom can be eliminated without loss of generality (but with some loss of convenience). Firstly, as in the case of encryption schemes, one may modify the key-generation algorithm so that on input 1^n it outputs a pair of n-bit long keys. Two more fundamental restrictions, which actually do not affect the existence of secure schemes, follow.

Randomization in the signing process: In contrast to the situation with respect to encryption schemes (see Sections 5.2 and 5.3), randomization is not essential to the actual signing and verifying processes (but is, as usual, essential to key generation). That is, without loss of generality (but with possible loss in efficiency), the signing algorithm may be *deterministic*, and in all schemes we present (in the current chapter) the verification algorithm is indeed deterministic. For details, see Exercise 1.

Canonical verification in the private-key version: As hinted above, in the private-key case, we may just identify the signing and verification keys (i.e., $k \stackrel{\rm def}{=} s = v$). Furthermore (following the comment about deterministic signing), without loss of generality, verification may amount to comparing the alleged signature against the one produced by the verification algorithm (as done by the signing algorithm). That is, we may let $V_k(\alpha,\beta) \stackrel{\rm def}{=} 1$ if and only if $\beta = S_k(\alpha)$. For details, see Exercise 2.

6.1.4.3 Weaker notions of security and some popular schemes

Weaker notion of security have been considered in the literature. The various notions refer to two parameters: (1) the type of attack, and (2) when is the adversary considered to be a success. Indeed, Definition 6.1.2 refers to the most severe type of attacks (i.e., unrestricted chosen message attacks) and to the most liberal notion of success (i.e., ability to produce a valid signature to any new message). The interested reader is referred to Section 6.6.5.

We note that *plain* RSA as well as plain versions of Rabin's scheme and the DSS are not secure under Definition 6.1.2. However, these schemes satisfy weaker notions of security, provided that some (standard) intractability assumptions hold. Furthermore, variants of these signature schemes (in which the function is not applied directly to the document itself) may be secure (under Definition 6.1.2).

6.2 Length-restricted signature scheme

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 yielding *secure* signatures only to documents of a certain predetermined length. The effect of the length-restriction is more dramatic here (in the context of signature schemes) than it was in the context of encryption schemes; compare the following to Section 5.3.2. Nevertheless, as we shall show (see Theorem 6.2.2 below), if the length restriction is not too low then the full power of signature schemes can be regained.

6.2.1 Definition

The essence of the length-restriction is in that security is guaranteed only with respect to documents of the predetermined length. Note that the question of what is the result of invoking the signature algorithm on a document of improper length is immaterial. What is important is that an attacker (of a length-restricted scheme) is deemed successful only if it produces a signature *to a* (different) *document of proper length.* Still, for sake of concreteness (and simplicity of subsequent treatment), we define the basic mechanism only for documents of proper length.

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* $\Pr[V(v, \alpha, S(s, \alpha)) = 1] = 1$.

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 stress that the essential modification is presented in the security condition is that considers an adversary to be successful only it case it forges a signature to a (different) document α of the proper length (i.e., $|\alpha| = \ell(n)$).

6.2.2 The power of length-restricted signature schemes

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 $2^{\ell(n)} = \text{poly}(n)$ possible documents).[3] In contrast, this triviality does not hold for public-key signature schemes. (For both claims, see Exercise 5.) On the other hand, in both (private-key and public-key) cases, 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.*

Results of the above flavor can be established in two different ways, corresponding to two methods of converting an ℓ-restricted signature scheme into a full-fledged one. Both methods are applicable both to private-key and public-key signature schemes. The first method (presented in Section 6.2.2.1) consists of parsing the original document into blocks (with proper "linkage" between blocks!),

[3] Recall, that such triviality does not hold in the context of encryption schemes; not even in the private-key case. See Section 5.3.2.

and applying the ℓ-restricted scheme to each block. The second method (presented in Section 6.2.2.2) 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. Thus, the second method requires an additional assumption (i.e., the existence of "collision-free" hashing), and so Theorem 6.2.2 (as stated) is actually proved using the first method. The second method is presented because it offers other benefits; in particular, it will play an important role in subsequent sections (e.g., in Sections 6.3.1.2 and 6.4.1.3).

6.2.2.1 Signing (augmented) blocks

In this subsection we present a simple method for constructing general signature schemes out of length-restricted ones, and doing so we establish Theorem 6.2.2. Loosely speaking, the method consists of parsing the original document into blocks (with proper "linkage" between blocks!), and applying the length-restricted scheme to each (augmented) block.

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 viewing documents as sequences of strings, each of length $\ell'(n) = \ell(n)/O(1)$. That is, we associate $\alpha = \alpha_1 \cdots \alpha_t$ with the sequence $(\alpha_1, ..., \alpha_t)$, where each α_i has length $\ell'(n)$. (At this point, the reader may think of $\ell'(n) = \ell(n)$, but actually we will use $\ell'(n) = \ell(n)/4$ in order to make room for further information.)

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 $(\alpha_1, \alpha_2, \alpha_3)$ can easily present a signature to the sequence (α_1, α_2). 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 identifier with each sequence and incorporating this identifier 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.*[4] *Next, S' uniformly selects $r \in \{0,1\}^{\ell'(n)}$. For $i = 1, ..., t$, algorithm S' computes*

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

where i and t are represented as $\ell'(n)$-bit long strings. 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 identifier r*". Finally, S' outputs as signature the sequence*

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

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 alleged 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 alleged 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, and holds for both the private-key and the public-key models.

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, A invokes A' with input identical to its own input (which is the security parameter or the verification-key depending on the model), and uses its own oracle 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 by 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 this pair in order to form a document-signature pair relative to the ℓ-restricted scheme, (G, S, V).

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

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 identifier supplied in the forged signature is different from the random identifiers 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 successful forgery with respect to the ℓ-restricted scheme.

2. The identifier supplied in the forged signature equals the random identifier supplied (by A) as part of *exactly one* of the signatures given to A'.

   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 different 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 (r,t,j,α_j) is different from each query $(r^{(i')},t^{(i')},j',\alpha_{j'}^{(i')})$ made by A to S_s.

  Justification for $(r,t,j,\alpha_j) \neq (r^{(i')},t^{(i')},j',\alpha_{j'}^{(i')})$. If $i' \neq i$ then (by the case hypothesis regarding uniqueness of i s.t. $r^{(i)} \neq r$) it holds that $r^{(i')} \neq r$. Otherwise (i.e., $i' = i$) either $j' \neq j$ or $\alpha_j \neq \alpha_{j'}^{(i')} = \alpha_j^{(i)}$.

  Thus, $((r,t,j,\alpha_j),\beta_j)$ is a valid document-signature pair with a document different than all queries made to S_s.

3. The identifier supplied in the forged signature equals the random identifiers 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 identifier. 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.[5] Thus, assuming that A' forges relative to the complex scheme with non-negligible probability $\varepsilon'(n)$, it follows that A forges relative to the length-restricted scheme with non-negligible probability $\varepsilon(n) \geq \varepsilon'(n) - \text{poly}(n) \cdot 2^{-\ell(n)/4}$, in contradiction to the proposition's hypothesis. ■

Comment: We call the reader's attention to the essential role of the hypothesis that ℓ is super-logarithmic in the proof of Proposition 6.2.4. Indeed, Construction 6.2.3 is insecure in case $\ell(n) = O(\log n)$. The reason being that, by asking for polynomially-many signatures, the adversary may obtain two S_s'-signatures that use the same (random) identifier. Furthermore, with some care, these signatures yield existential forgery (see Exercise 6).

[5] 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 probability $1/\text{poly}(n)$ because there are only $\text{poly}(n)$-many possibilities).

6.2.2.2 Signing a hash value

In this subsection we present an alternative method for constructing general signature schemes out of length-restricted ones. Loosely speaking, the method consists of hashing the document into a short (fixed-length) string (via an adequate hashing scheme), and applying the length-restricted signature scheme to the resulting hash-value. This two-stage process is referred to as the *hash and sign* paradigm.

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.

Collision-free hashing functions. 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 a random s it is hard to find $x \neq x'$ such 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):[6] *For some polynomial p, all sufficiently large n's and every s in the range of $I(1^n)$ it holds that $n \leq p(|s|)$. Furthermore, n can be computed in polynomial-time from s.*
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,*

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

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

[6] This condition is made merely to avoid annoying technicalities. In particular, it allows the collision-forming adversary to run for poly(n)-time (since by this condition poly(n) = poly$(|s|)$) as well as allows to determine n from s. Note that $|s| = \text{poly}(n)$ holds by definition of I.

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 $2^{\ell(n)}+1$ different preimages and computing their 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 alleged 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 reach a contradiction. We start with the description of algorithm A, which is designed to attack the ℓ-restricted scheme (G,S,V). We stress that almost the same description applies both in the private-key and public-key case.

On input x, which equals the security parameter 1^n in the private-key case and a verification-key v otherwise (i.e., in the public-key case), the adversary

A operates as follows. First A 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, A invokes A' (on input 1^n or (r, v) depending on the case), and uses r as well as its own oracle S_s in order to emulate the oracle $S'_{r,s}$ for A'. The emulation is done in a straightforward manner; that is, algorithm A will act as $S'_{r,s}$ does by using the oracle S_s (i.e., to answer query q, algorithm A makes the query $h_r(q)$). 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). That is, if A' outputs the document-signature pair (α, β), then A will output the document-signature pair $(h_r(\alpha), \beta)$.

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 two cases regarding the forging event, letting $(\alpha^{(i)}, \beta^{(i)})$ 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):

Case 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.

Case 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 $(\alpha, \alpha^{(i)})$ forms a collision under h_r (and we do *not* obtain success in existential forgery relative to the ℓ-restricted scheme).

Thus, if Case 1 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 Case 2 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, denoted 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 (given to it as input). Recall that A, in turn, emulates an attack of A' on the signing oracle $S'_{r,s}$, and that A answers the query q' made by A' by forwarding the query $q = h_r(q')$ to S_s. Thus, B

actually emulates the attack of A' (on the signing oracle $S'_{r,s}$), and does so in a straightforward manner; that is, to answer query q' made by A', algorithm B first obtains $q = h_r(q')$ (using its knowledge of r) and then answers with $S_s(q)$ (using its knowledge of s). Finally, when A' outputs a forged document-signature pair, algorithm B checks whether Case 2 occurs (i.e., whether $h_r(\alpha) = h_r(\alpha^{(i)})$ holds for some i), in which case it obtains (and outputs) a collision under h_r. (Note that in the public-key case B invokes A' on input (r, v), whereas in the private-key case B invokes A' on input 1^n. Thus, in the private-key case, B actually does not use r but rather an oracle access to 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. ■

Comment: For the private-key case, the proof of Proposition 6.2.7 actually established a stronger claim than stated. The proof holds even for a weaker definition of collision-free hashing in which the adversary is not given a description of the hashing function, but can rather obtain its values at any preimage of its choice. This observation is further pursued in Section 6.3.1.3.

On using the hash and sign paradigm in practice. The *hash-and-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 one way of constructing 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 or not it is collision-free is a very dangerous practice.

6.2.3 * Constructing collision-free hashing functions

In view of the relevance of collision-free hashing to signature schemes, we now take a small detour from the main topic and consider the construction of collision-free hashing. We show how to construct collision-free hashing functions using a claw-free collection of permutations, and how restricted notions of collision-free hashing may be used to obtain full-fledged collision-free hashing.

6.2.3.1 A construction based on claw-free permutations

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

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

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 $x_1 x_2 \cdots x_m$ by $x_1 x_1 x_2 x_2 \cdots x_m x_m 01$.

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.*

Actually, as will become evident from the proof of Proposition 6.2.9, we do not need an algorithm that given an index s generates a uniformly distributed element in D_s; any efficient algorithm that generates elements in D_s (under any distribution) will do.

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,[7] 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

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

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

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

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. ■

6.2.3.2 Collision-free hashing via block-chaining

In this subsection we show how a restricted type of collision-free hashing (CFH) can be used to obtain full-fledge collision-free hashing (CFH). Specifically, we refer to the following restriction of Definition 6.2.5.

[7] 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.

Definition 6.2.10 (length-restricted collision-free hashing functions): *Let $\ell', \ell : \mathbb{N} \to \mathbb{N}$. A collection of functions $\{h_s : \{0,1\}^{\ell'(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is called ℓ'-*restricted collision-free hashing *if there exists a probabilistic polynomial-time algorithm I such that the following holds*

1. (admissible indexing – technical): *As in Definition 6.2.5.*
2. (efficient evaluation): *There exists a polynomial-time algorithm that given s and $x \in \{0,1\}^{\ell'(|s|)}$, returns $h_s(x)$.*
3. (hard to form collisions): *As in Definition 6.2.5, 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 pair in $\{0,1\}^{\ell'(|s|)} \times \{0,1\}^{\ell'(|s|)}$ that forms 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,*

$$\mathsf{Pr}\left[A(I(1^n)) \in \{0,1\}^{2\cdot\ell'(|I(1^n)|)} \text{ is a collision under } h_{I(1^n)}\right] < \frac{1}{p(n)}$$

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

Indeed, we focus on the case $\ell'(n) = \text{poly}(n)$, or else the hardness condition holds vacuously (since no polynomial-time algorithm can print a pair of strings of super-polynomial length). On the other hand, we only care about the case $\ell'(n) > \ell(n)$ (or else the functions may be 1-1). Finally, recall that ℓ must be super-logarithmic.

Construction 6.2.11 (from 2ℓ-restricted CFH to full-fledged CFH): *Let $\{h'_s : \{0,1\}^{2\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ be a collection of functions. Consider the collection $\{h_s : \{0,1\}^* \to \{0,1\}^{2\ell(|s|)}\}_{s\in\{0,1\}^*}$, where $h_s(x)$ is defined by the following process, which we call* block chaining*:*

1. *Break x into $t \stackrel{\text{def}}{=} \lceil |x|/\ell(|s|)\rceil$ consecutive blocks, while possibly padding the last block with 0's, such that each block has length $\ell(|s|)$. Denote these $\ell(|s|)$-bit long blocks by $x_1, ..., x_t$. That is, $x_1 \cdots x_t = x0^{t\cdot\ell(|s|)-|x|}$.*

   For sake of uniformity, in case $|x| \leq \ell(|s|)$, we let $t = 2$ and $x_1x_2 = x0^{2\ell(|s|)-|x|}$. On the other hand, we may assume that $|x| < 2^{\ell(|s|)}$, and so $|x|$ can be represented by an $\ell(|s|)$-bit long string.[8]
2. *Let $y_1 \stackrel{\text{def}}{=} x_1$. For $i = 2, ..., t$, compute $y_i = h'_s(y_{i-1}x_i)$.*
3. *Set $h_s(x)$ to equal $(y_t, |x|)$.*

[8] The adversary trying to form collisions with respect to h_s runs in poly($|s|$)-time. Using $\ell(|s|) = \omega(\log |s|)$, it follows that such an adversary cannot output a string of length $2^{\ell(|s|)}$. (The same holds, of course, also for legitimate usage of the hashing function.)

An interesting property of Construction 6.2.11 is that it allows to compute the hash-value of an input string while processing the input in an on-line fashion; that is, the implementation of the hashing process may process the input x in a block-by-block manner, while storing only the current block and a small amount of state information (i.e., the current y_i and the number of blocks encountered so far). This property is important in applications in which one wishes to hash a long stream of input bits.

Proposition 6.2.12 *Let* $\{h'_s : \{0,1\}^{2\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ *and* $\{h_s : \{0,1\}^* \to \{0,1\}^{2\ell(|s|)}\}_{s\in\{0,1\}^*}$ *be as in Construction 6.2.11, and suppose that the former is a collection of* 2ℓ*-restricted collision-free hashing functions. Then the latter constitute a* (full fledged) *collection of collision-free hashing functions.*

Proof: Forming a collision under h_s means finding $x \neq x'$ such that $h_s(x) = h_s(x')$. By the definition of h_s, this means that $(y_t, |x|) = h_s(x) = h_s(x') = (y'_{t'}, |x'|)$, where t, t' and $y_t, y'_{t'}$ are determined by $h_s(x)$ and $h_s(x')$. In particular, it follows that $|x| = |x'|$ and so $t = t'$ (where, except when $|x| \leq \ell(|s|)$, it holds that $t = \lceil |x|/\ell(|s|) \rceil = \lceil |x'|/\ell(|s|) \rceil = t'$). Recall that $y_t = y'_t$ and consider two cases:

Case 1: If $(y_{t-1}, x_t) \neq (y'_{t-1}, x'_t)$ then we obtain a collision under h'_s (since $h'_s(y_{t-1}x_t) = y_t = y'_t = h'_s(y'_{t-1}x'_t)$), and derive a contradiction to its collision-free hypothesis.

Case 2: Otherwise $(y_{t-1}, x_t) = (y'_{t-1}, x'_t)$, and we consider the two corresponding cases with respect to the relation of (y_{t-2}, x_{t-1}) to (y'_{t-2}, x'_{t-1}).

Eventually, since $x \neq x'$, we get to a situation in which $y_i = y'_i$ and $(y_{i-1}, x_i) \neq (y'_{i-1}, x'_i)$, which is handled as in the first case.

We now provide a formal implementation of the above intuitive argument. Suppose towards the contradiction that there exist a probabilistic polynomial-time algorithm A that on input s attempts to forms a collision under h_s. Then, we construct an algorithm that will, with similar probability, succeeds to form a suitable (i.e., length restricted) collision under h'_s. Algorithm $A'(s)$ operates as follows:

1. Invokes $A(s)$ and obtains $(x, x') \leftarrow A(s)$.

   If $h_s(x) \neq h_s(x')$ then A failed, and A' halts without output. In the sequel, we assume that $h_s(x) = h_s(x')$.

2. $A'(s)$ computes $t, x_1, ..., x_t$ and $y_1, ..., y_t$ (resp., $t', x'_1, ..., x'_t$ and $y'_1, ..., y'_t$) as in Construction 6.2.11. Note that (since $h_s(x) = h_s(x')$) it holds that $t = t'$ and $y_t = y'_t$. Next, $A'(s)$ determines $i \in \{2, ..., t\}$ such that $y_i = y'_i$ and $(y_{i-1}, x_i) \neq (y'_{i-1}, x'_i)$, and outputs the pair $(y_{i-1}x_i, y'_{i-1}x'_i)$

   As argued above and elaborated below, such an i must exist, and the output forms a collision under h'_s (because $h'_s(y_{i-1}x_i) = y_i = y'_i = h'_s(y'_{i-1}x'_i)$ and $y_{i-1}x_i \neq y'_{i-1}x'_i$).

Pending on the existence of a suitable i, whenever $A(s)$ forms a collision under h_s, it holds that $A'(s)$ outputs a pair of $2\ell(s)$-bit long strings that form a collision under h'_s, and so the proposition follows. Thus, it is left to prove the existence of a suitable i (i.e., an i such that $y_i = y'_i$ and $(y_{i-1}, x_i) \neq (y'_{i-1}, x'_i)$).

> On the existence of a suitable i: Starting with $j = t$ and decrementing j at each step, we prove that either the current j is suitable (i.e., $y_j = y'_j$ and $(y_{j-1}, x_j) \neq (y'_{j-1}, x'_j)$) or both $y_{j-1} = y'_{j-1}$ and $x_1 \cdots x_{j-1} \neq x'_1 \cdots x'_{j-1}$. This claim certainly holds for $j = t$, because $y_t = y'_t$ and $x_1 \cdots x_t = x0^{t\ell(|s|)-|x|} \neq x'0^{t\ell(|s|)-|x|} = x'_1 \cdots x'_t$ (which implies that either $(y_{t-1}, x_t) \neq (y'_{t-1}, x'_t)$ or both $y_{t-1} = y'_{t-1}$ and $x_1 \cdots x_{t-1} \neq x'_1 \cdots x'_{t-1}$). More generally, suppose that $y_j = y'_j$ and $x_1 \cdots x_j \neq x'_1 \cdots x'_j$, then either j is suitable (i.e., $(y_{j-1}, x_j) \neq (y'_{j-1}, x'_j)$) or $(y_{j-1}, x_j) = (y'_{j-1}, x'_j)$, which implies that both $y_{j-1} = y'_{j-1}$ and $x_1 \cdots x_{j-1} \neq x'_1 \cdots x'_{j-1}$. It follows that some i must be suitable (or else for $j = 1$ we have $x_1 \cdots x_{j-1} \neq x'_1 \cdots x'_{j-1}$, which is impossible).

The proposition follows. ■

6.2.3.3 Collision-free hashing via tree-hashing

Using 2ℓ-restricted collision-free hashing functions, we now present an alternative construction of (full fledged) collision-free hashing functions. The alternative construction will have the extra property of supporting verification of a bit in the input (with respect to the hash value) within complexity that is independent of the length of the input (see below).

Construction 6.2.13 (from 2ℓ-restricted CFH to full-fledged CFH – an alternative construction): *Let $\{h'_s : \{0,1\}^{2\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ be a collection of functions. Consider the collection $\{h_s : \{0,1\}^* \to \{0,1\}^{2\ell(|s|)}\}_{s\in\{0,1\}^*}$, where $h_s(x)$ is defined by the following process, called* tree hashing*:*

1. *Break x into $t \stackrel{\rm def}{=} 2^{\lceil \log_2(|x|/\ell(|s|)) \rceil}$ consecutive blocks, while possibly adding dummy 0-blocks and padding the last block with 0's, such that each block has length $\ell(|s|)$. Denote these $\ell(|s|)$-bit long blocks by $x_1, ..., x_t$. That is, $x_1 \cdots x_t = x0^{t\cdot\ell(|s|)-|x|}$.*

   Let $d = \log_2 t$, and note that d is a positive integer.

   Again, for sake of uniformity, in case $|x| \leq \ell(|s|)$, we let $t = 2$ and $x_1 x_2 = x0^{2\ell(|s|)-|x|}$. On the other hand, again, we assume that $|x| < 2^{\ell(|s|)}$, and so $|x|$ can be represented by an $\ell(|s|)$-bit long string.

2. *Let $i = 1, ..., t$, let $y_{d,i} \stackrel{\rm def}{=} x_i$.*

3. *For $j = d-1, ..., 1, 0$ and $i = 1, ..., 2^j$, compute $y_{j,i} = h'_s(y_{j+1,2i-1} y_{j+1,2i})$.*

4. *Set $h_s(x)$ to equal $(y_{0,1}, |x|)$.*

That is, hashing is performed by placing the $\ell(|s|)$-bit long blocks of x at the leaves of a binary tree of depth d, and computing the values of internal nodes by applying h'_s to the values associated with the two children (of the node). The final hash-value consists of the value associated with the root (i.e., the only level-0 node) and the length of x.

Proposition 6.2.14 *Let* $\{h'_s : \{0,1\}^{2\ell(|s|)} \to \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ *and* $\{h_s : \{0,1\}^* \to \{0,1\}^{2\ell(|s|)}\}_{s\in\{0,1\}^*}$ *be as in Construction 6.2.13, and suppose that the former is a collection of* 2ℓ*-restricted collision-free hashing functions. Then the latter constitute a* (full fledged) *collection of collision-free hashing functions.*

Proof Sketch: Forming a collision under h_s means finding $x \neq x'$ such that $h_s(x) = h_s(x')$. By the definition of h_s, this means that $(y_{0,1}, |x|) = h_s(x) = h_s(x') = (y'_{0,1}, |x'|)$, where (t, d, t', d'), $y_{0,1}$ and $y'_{0,1}$ are determined by $h_s(x)$ and $h_s(x')$. In particular, it follows that $|x| = |x'|$ and so $d = d'$ (since $2^d = t = t' = 2^{d'}$). Recall that $y_{0,1} = y'_{0,1}$, and let us state this fact by saying that for $j = 0$ and for every $i \in \{1, ..., 2^j\}$ it holds that $y_{j,i} = y'_{j,i}$. Starting with $j = 0$, we consider two cases (for level $j + 1$ in the tree):

Case 1: If for some $i \in \{1, ..., 2^{j+1}\}$ it holds that $y_{j+1,i} \neq y'_{j+1,i}$ then we obtain a collision under h'_s, and derive a contradiction to its collision-free hypothesis. Specifically, the collision is obtained because $z \stackrel{\rm def}{=} y_{j+1,2\lceil i/2\rceil-1}y_{j+1,2\lceil i/2\rceil}$ is different from $z' \stackrel{\rm def}{=} y'_{j+1,2\lceil i/2\rceil-1}y'_{j+1,2\lceil i/2\rceil}$, whereas $h'_s(z) = y_{j,\lceil i/2\rceil} = y'_{j,\lceil i/2\rceil} = h(z')$.

Case 2: Otherwise for every $i \in \{1, ..., 2^{j+1}\}$ it holds that $y_{j+1,i} = y'_{j+1,i}$. In this case, we consider the next level.

Eventually, since $x \neq x'$, we get to a situation in which for some $j \in \{1, ..., d-1\}$ and some $i \in \{1, ..., 2^{j+1}\}$ it holds that $z \stackrel{\rm def}{=} y_{j+1,2\lceil i/2\rceil-1}y_{j+1,2\lceil i/2\rceil}$ is different from $z' \stackrel{\rm def}{=} y'_{j+1,2\lceil i/2\rceil-1}y'_{j+1,2\lceil i/2\rceil}$, whereas $h'_s(z) = y_{j,\lceil i/2\rceil} = y'_{j,\lceil i/2\rceil} = h(z')$. This situation is handled as in the first case.

The actual argument proceeds as in the proof of Proposition 6.2.12. ■

A local verification property. Construction 6.2.13 has the extra property of supporting efficient verification of bits in x with respect to the hash value. That is, suppose that for a randomly selected h_s, one party holds x and the other party holds $h_s(x)$. Then, for every i, the first party may provide a short (efficiently verifiable) certificate that x_i is indeed the $i^{\rm th}$ block of x. The certificate consists of the sequence of pairs $(y_{d,2\lceil i/2\rceil-1}, y_{d,2\lceil i/2\rceil}), ..., (y_{1,2\lceil i/2^d\rceil-1}, y_{1,2\lceil i/2^d\rceil})$, where d and the $y_{j,k}$'s are computed as in Construction 6.2.13 (and $(y_{0,1}, |x|) = h_s(x)$). The certificate is verified by checking whether or not $y_{j-1,\lceil i/2^{d-j+1}\rceil} = h'_s(y_{j,2\lceil i/2^{d-j+1}\rceil-1}y_{j,2\lceil i/2^{d-j+1}\rceil})$, for every $j \in \{1, ..., d\}$. Note that if the first party can present two different values for the $i^{\rm th}$ block of x along with corresponding certificates then it can also form collisions under h'_s. Construction 6.2.13 and its local-verification property were already used in this work (i.e., in the construction of highly-efficient argument systems, presented in Section 4.8.4). Finally, we note the similarity between the local-verification property of Construction 6.2.13 and the authentication-tree of Section 6.4.2.2.

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 such a scheme by the popular abbreviation MAC (which actually abbreviates the more traditional term of a Message Authentication Code).

6.3.1 Applying a pseudorandom function to the document

A scheme for message authentication can be obtained by applying a pseudorandom function (specified by the key) to the message (which one wishes to authenticate). The simplest implementation of this idea is presented in Section 6.3.1.1, whereas more sophisticated implementations are presented in Sections 6.3.1.2 and 6.3.1.3.

6.3.1.1 A simple construction and a plausibility result

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 super-logarithmically growing ℓ. Indeed, this is our starting point.

Construction 6.3.1 (an ℓ-restricted MAC based on pseudorandom functions): *Let ℓ be a super-logarithmically growing function, and $\{f_s : \{0,1\}^{\ell(|s|)} \rightarrow \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ 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.)

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 alleged signature β, we accept if and only if $\beta = f_s(\alpha)$.*

Indeed, signing amounts to applying f_s to the given document string, and verification amounts to comparing a given value to the result of applying f_s to the document. Analogous constructions can be presented by using the generalized notions of pseudorandom functions defined in Definitions 3.6.9 and 3.6.12 (see further comments in the following subsections). In particular, using a pseudorandom function ensemble of the form $\{f_s : \{0,1\}^* \rightarrow \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, we obtain a general message authentication scheme (rather than a length-restricted one). Below, we only prove 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 7.)

Proposition 6.3.2 *Suppose that $\{f_s : \{0,1\}^{\ell(|s|)} \rightarrow \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$ is a pseudorandom function, and that ℓ is a super-logarithmically 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 truly 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 $\{f_s : \{0,1\}^{\ell(|s|)} \rightarrow \{0,1\}^{\ell(|s|)}\}_{s\in\{0,1\}^*}$. 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. *Machine A attacks the actual scheme*: On input 1^n, machine A is given oracle access to (the signing process) $f_s : \{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{\ell(n)}$, 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. A is deem successful if and only if $\beta = f_s(\alpha)$.

2. *Machine A attacks the ideal scheme*: On input 1^n, machine A is given oracle access to a function $F : \{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{\ell(n)}$, 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. Again, A 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 $f : \{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{\ell(n)}$, 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 $\{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{\ell(n)}$ 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 8. Thus, we have:

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

In contrast the the feasibility result stated in Theorem 6.3.3, we now present alternative ways of using pseudorandom functions to obtain secure message authentication schemes (MACs). These alternatives yield more efficient schemes, where efficiency is measures it terms of the *length of the signatures* and the *time* it takes to produce and verify them.

6.3.1.2 * Using the hash-and-sign paradigm

Theorem 6.3.3 was proved by combining the length-restricted MAC of Construction 6.3.1 with the simple but wasteful idea of providing signatures (authentication tags) for each block of the document (i.e., Construction 6.2.3). In particular, the signature produced this way is longer than the document. Instead, here we suggest to use the second method of converting length-restricted MACs into full-fledged ones; that is, the hash-and-sign method of Construction 6.2.6. This will yield signatures of a fixed length (i.e., independent of the length of the document). Combining the hash-and-sign method with a length-restricted MAC of Construction 6.3.1 (which is based on pseudorandom functions), we obtain the following construction.

Construction 6.3.4 (hash and sign using pseudorandom functions): *Let $\{f_s : \{0,1\}^{|s|} \rightarrow \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$ be a pseudorandom function ensemble and $\{h_r : \{0,1\}^* \rightarrow \{0,1\}^{|r|}\}_{r\in\{0,1\}^*}$ be a collection of collision-free hashing functions. Furthermore, for simplicity we assume that, when invoked on input 1^n, the indexing algorithm I of the collision-free hashing collection outputs an n-bit long index. The general message authentication scheme, (G, S, V), is as follows:*

key-generation with G: *On input 1^n, algorithm G selects uniformly $s \in \{0,1\}^n$, and invokes the indexing algorithm I to obtain $r \leftarrow I(1^n)$. The key-pair output by G is $((r,s),(r,s))$.*

signing with S: *On input a signing-key (r,s) in the range of $G_1(1^n)$ and a document $\alpha \in \{0,1\}^*$, algorithm S outputs the signature/tag $f_s(h_r(\alpha))$.*

verification with V: *On input a verification-key (r,s) in the range of $G_2(1^n)$, a document $\alpha \in \{0,1\}^*$, and a alleged signature β, algorithm outputs 1 if and only if $f_s(h_r(\alpha)) = \beta$.*

Combining Propositions 6.2.7 and 6.3.2, it follows that *Construction 6.3.4 constitutes a secure message authentication scheme* (MAC), provided that the ingredients are as postulated. In particular, this means that Construction 6.3.4 yields a secure MAC, provided that collision-free hashing functions exist (and are used in Construction 6.3.4). While this result uses a seemingly stronger assumption than the existence of one-way functions (used to establish the Theorem 6.3.3), it yields more efficient MACs both in terms of signature length (as discussed above) and authentication time (to be discussed next).

Construction 6.3.4 yields faster signing and verification algorithms than the construction resulting from combining Constructions 6.2.3 and 6.3.1, provided that hashing a long string is less time-consuming than applying a pseudorandom function to it (or to all its blocks). The latter assumption is consistent with the current state-of-art regarding the implementation of both primitives. Further speed improvements are discussed in Section 6.3.1.3.

An alternative presentation: Construction 6.3.4 was analyzed by invoking the hash-and-sign paradigm (i.e., Proposition 6.2.7), while referring to the fixed-length MAC arising from the pseudorandom function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$. An alternative analysis may proceed by first establishing that $\{g_{s,r} = f_s \circ h_r\}_{s\in\{0,1\}^*, r\leftarrow I(1^{|s|})}$ is a generalized pseudorandom function (as in Definition 3.6.12), and next observing that any such ensemble yields a full-fledged MAC (see Exercise 7).

6.3.1.3 * A variation on the hash-and-sign paradigm or using non-cryptographic hashing plus hiding

Construction 6.3.4 combines the use of a collision-free hashing function with the application of a pseudorandom function. Here we take another step towards speeding-up message authentication by showing that the collision-free hashing can be replaced with *ordinary* (i.e., non-cryptographic) *hashing*, provided that a pseudorandom function is applied to the result. Before getting into details, let us explain why we can use non-cryptographic hashing and why this may lead to efficiency improvements.

- Since we are in the private-key setting, the adversary does not get the description of the hash function used in the hash-and-sign process. Furthermore, applying the pseudorandom function to the hash-value hides it from the adversary. Thus, when trying to form collisions under the hash function, the adversary is in "total darkness" and may only rely on the collision probability of the hashing function (as defined below). (Recall that in case the adversary fails to form collision, it must succeed in forging with respect to the length-restricted scheme if it wishes to forge with respect to the full-fledged scheme.)

- The reason that applying an ordinary hashing, rather than a collision-free hash function, may yield an efficiency improvement is that the former may be more efficient than the latter. This is to be expected given that ordinary hashing needs only satisfy a weak (probabilistic) condition, whereas collision-free hashing refers to a more complicated (intractability) condition.[9]

By *ordinary hashing* we mean function ensembles as defined in Section 3.5.1.1. For starters, recall that these are collections of functions mapping $\ell(n)$-bit strings to $m(n)$-bit strings. These collections are associated with a set of strings, denoted $S_{\ell(n)}^{m(n)}$, and we may assume that $S_{\ell(n)}^{m(n)} \equiv \{0,1\}^n$. Specifically, we call $\{S_{\ell(n)}^{m(n)}\}_{n\in\mathbb{N}}$ a hashing ensemble if it satisfies the following three conditions:

1. *Succinctness*: $n = \mathrm{poly}(\ell(n) + m(n))$.

2. *Efficient evaluation*: there exists a polynomial-time algorithm that, on input a representation of a function, h (in $S_{\ell(n)}^{m(n)}$), and a string $x \in \{0,1\}^{\ell(n)}$, returns $h(x)$.

3. *Pairwise independence*: for every $x \neq y \in \{0,1\}^{\ell(n)}$, if h is uniformly selected in $S_{\ell(n)}^{m(n)}$ then $h(x)$ and $h(y)$ are independent and uniformly distributed in $\{0,1\}^{m(n)}$. That is, for every $\alpha, \beta \in \{0,1\}^{m(n)}$,
$$\mathsf{Pr}_h[h(x) = \alpha \ \wedge \ h(y) = \beta] \;\; = \;\; 2^{-2m(n)}$$

In fact, for the current application, we can replace the third condition by the following weaker condition, parameterized by a function $\mathsf{cp} : \mathbb{N} \to [0,1]$ (s.t. $\mathsf{cp}(n) \geq 2^{-m(n)}$): for every $x \neq y \in \{0,1\}^{\ell(n)}$,
$$\mathsf{Pr}_h[h(x) = h(y)] \;\; \leq \;\; \mathsf{cp}(n) \tag{6.3}$$

Indeed, the pairwise independence condition implies that Eq. (6.3) is satisfied with $\mathsf{cp}(n) = 2^{-m(n)}$. Note that Eq. (6.3) asserts that the collision probability of $S_{\ell(n)}^{m(n)}$ is at most $\mathsf{cp}(n)$, where the collision probability refers to the probability that $h(x) = h(y)$ when h is uniformly selected in $S_{\ell(n)}^{m(n)}$ and $x \neq y \in \{0,1\}^{\ell(n)}$ are arbitrary fixed strings.

Hashing ensembles with $n \leq \ell(n) + m(n)$ and $\mathsf{cp}(n) = 2^{-m(n)}$ can be constructed (for a variety of functions $\ell, m : \mathbb{N} \to \mathbb{N}$, e.g., $\ell(n) = 2n/3$ and $m(n) = n/3$); see Exercise 18. Using such ensembles, we first present a construction of length-restricted message authentication schemes.

[9] This intuition may not hold when comparing a construction of ordinary hashing that is rigorously analyzed with an ad-hoc suggestion of a collision-free hashing. But it certainly holds when comparing the former to the constructions of collision-free hashing that are based on a well-established intractability assumption.

Construction 6.3.5 (Construction 6.3.4, revisited – length-restricted version): *Let* $\{h_r : \{0,1\}^{\ell(|r|)} \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^*}$ *and* $\{f_s : \{0,1\}^{m(|s|)} \to \{0,1\}^{m(|s|)}\}_{s\in\{0,1\}^*}$ *be efficiently computable function ensembles. We construct the following* ℓ*-restricted scheme,* (G, S, V)*:*

key-generation with G: *On input* 1^n*, algorithm* G *selects independently and uniformly* $r, s \in \{0,1\}^n$*. The key-pair output by* G *is* $((r, s), (r, s))$*.*

signing with S: *On input a signing-key* (r, s) *in the range of* $G_1(1^n)$ *and a document* $\alpha \in \{0,1\}^{\ell(n)}$*, algorithm* S *outputs the signature/tag* $f_s(h_r(\alpha))$*.*

verification with V: *On input a verifying-key* (r, s) *in the range of* $G_2(1^n)$*, a document* $\alpha \in \{0,1\}^{\ell(n)}$*, and a alleged signature* β*, algorithm outputs 1 if and only if* $f_s(h_r(\alpha)) = \beta$*.*

Proposition 6.3.6 *Suppose that* $\{f_s : \{0,1\}^{m(|s|)} \to \{0,1\}^{m(|s|)}\}_{s\in\{0,1\}^*}$ *is a pseudorandom function, and that the collision probability of the collection* $\{h_r : \{0,1\}^{\ell(|r|)} \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^*}$ *is a negligible function of* $|r|$*. Then Construction 6.3.5 constitutes a secure* ℓ*-restricted message authentication scheme.*

In particular, the second hypothesis implies that $2^{-m(n)}$ is a negligible function in n. By the above discussion, adequate collections of hashing functions exists for $\ell(n) = 2n/3$ (and $m(n) = n/3$). We comment that, under the above hypothesis, the collection $\{g_{s,r} : f_s \circ h_r\}_{|s|=|r|}$ constitutes a pseudorandom function ensemble: This is implicitly shown in the following proof, and is related to Exercise 31 in Chapter 3.

Proof Sketch: As in the proof of Proposition 6.3.2, we first consider the security of an ideal scheme in which the pseudorandom function is replaced by a truly random function (mapping $m(n)$-bit long strings to $m(n)$-bit long strings). Consider any (probabilistic polynomial-time) adversary attacking the *ideal scheme*. Such an adversary may obtain the signatures to polynomially-many $\ell(n)$-bit long strings of its choice. However, except with negligible probability, these strings are hashed to different $m(n)$-bit long strings, which in turn are mapped by the random function to totally independent and uniformly distributed $m(n)$-bit long strings. Furthermore, except with negligible probability, the $\ell(n)$-bit long string α contained in the adversary's (alleged message-signature) output pair is hashed to an $m(n)$-bit long string that is different from all the previous hash-values, and so the single valid signature corresponding to α is a uniformly distributed $m(n)$-bit long string that is independent of all previously seen signatures.

> On the distribution of signatures in the ideal scheme: Suppose that the hashing collection $\{h_r : \{0,1\}^{\ell(|r|)} \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^n}$ has collision probability $\mathsf{cp}(n)$, and $F : \{0,1\}^{m(n)} \to \{0,1\}^{m(n)}$ is a random function. Then, we claim that an adversary that obtains signatures to $t(n) - 1$ strings of its choice, succeeds in forging a signature to a new string with probability at most $t(n)^2 \cdot \mathsf{cp}(n) + 2^{-m(n)}$, regardless of its computational powers. The claim is proved by showing that, except with probability at most $t(n)^2 \cdot \mathsf{cp}(n)$, the $t(n)$ strings selected by the adversary are mapped

by h_r to distinct strings. The latter claim is proved by induction on the number of selected strings, denoted i, where the base case (i.e., $i = 1$) holds vacuously. Let $s_1, ..., s_i$ denote the strings selected so far, and recall that with probability at least $1 - i^2 \cdot \mathsf{cp}(n)$ the i hash-values $h_r(s_j)$'s are distinct. The adversary only sees the corresponding $F(h_r(s_j))$'s, which are uniformly and independently distributed (in a way independent of the values of the $h_r(s_j)$'s). Thus, loosely speaking, the adversary's selection of the next string, denoted s_{i+1}, is independent of the values of the $h_r(s_j)$'s, and so a collision of $h_r(s_{i+1})$ with one of the previous $h_r(s_j)$'s occurs with probability at most $i \cdot \mathsf{cp}(n)$. The induction step follows (since $1 - i^2 \cdot \mathsf{cp}(n) - i \cdot \mathsf{cp}(n) < 1 - (i+1)^2 \cdot \mathsf{cp}(n)$).

It follows that any adversary attacking the *ideal scheme* may succeed in existential forgery with at most negligible probability (provided it makes at most polynomially-many queries). 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 $\{f_s : \{0,1\}^{m(|s|)} \to \{0,1\}^{m(|s|)}\}_{s\in\{0,1\}^*}$. The exact implementation of the above argument follows the details given in the proof of Proposition 6.3.2. ■

Obtaining full-fledged MACs. Construction 6.3.5 can be generalized to obtain full-fledged MACs by using generalized hashing families that map arbitrary strings (rather than fixed-length ones) to fixed length strings. Specifically, for $\ell : \mathbb{N} \to \mathbb{N}$ and $\mathsf{cp} : \mathbb{N} \to [0,1]$, we call $\{h_r : \{0,1\}^* \to \{0,1\}^{m(|r|)}\}_{n\in\mathbb{N}}$ a generalized hashing ensemble with a (ℓ, cp)-collision property if it satisfies the following two conditions:

1. *Efficient evaluation*: there exists a polynomial-time algorithm that, on input r (representing the function h_r) and a string $x \in \{0,1\}^*$, returns $h_r(x)$.
2. *Collision probability*:[10] For every $n \in \mathbb{N}$ and $x \neq y$ such that $|x|, |y| \leq \ell(n)$, the probability that $h_r(x) = h_r(y)$ when r is uniformly selected in $\{0,1\}^n$ is at most $\mathsf{cp}(n)$.

For our construction of a full-fledged MAC, we need a generalized hashing ensemble with a (ℓ, cp)-collision property for some super-polynomial $\ell(n)$ and negligible $\mathsf{cp}(n)$ (e.g., $\ell(n) = 1/\mathsf{cp}(n) = 2^{-\varepsilon n^{\varepsilon}}$ for some constant $\varepsilon > 0$). The existence of such ensembles will be discussed below.

Proposition 6.3.7 (Construction 6.3.4, revisited – full-fledged version): *Suppose that* $\{f_s : \{0,1\}^{m(|s|)} \to \{0,1\}^{m(|s|)}\}_{s\in\{0,1\}^*}$ *is a pseudorandom function ensemble. For some super-polynomial* $\ell : \mathbb{N} \to \mathbb{N}$ *and negligible* $\mathsf{cp} : \mathbb{N} \to [0,1]$, *suppose that* $\{h_r : \{0,1\}^* \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^*}$ *is a generalized hashing ensemble with a* (ℓ, cp)*-collision property. Then the following* (G, S, V) *constitute a secure MAC:*

[10] Note that it is essential to restrict the collision condition to strings of bounded length. In contrast, for every finite family of functions H, there exists two different strings that are mapped to the same image by each function in H. For details, see Exercise 17.

key-generation with G: *On input 1^n, algorithm G selects independently and uniformly $r, s \in \{0,1\}^n$, and outputs $((r,s),(r,s))$.*

signing with S: *On input a signing-key (r,s) and a document $\alpha \in \{0,1\}^*$, algorithm S outputs the signature/tag $f_s(h_r(\alpha))$.*

verification with V: *On input a verifying-key (r,s), a document $\alpha \in \{0,1\}^{\ell(n)}$, and a alleged signature β, algorithm outputs 1 if and only if $f_s(h_r(\alpha)) = \beta$.*

Proof Sketch: The proof is identical to the proof of Proposition 6.3.6, except that here the (polynomial-time) adversary attacking the scheme may query for the signatures of strings of various lengths. Still, all these queries (as well as the final output) are of polynomial length and thus shorter than $\ell(n)$. Thus, the (ℓ, cp)-collision property implies that, except with negligible probability, all these queries (as well as the relevant part of the output) are hashed to different values. ■

On constructing adequate hashing ensembles. For some $\varepsilon > 0$ and $f(n) = 2^{\varepsilon n^\varepsilon}$, generalized hashing ensembles with a $(f, 1/f)$-collision property can be constructed is several ways. One way is by applying a tree-hashing scheme as in Construction 6.2.13; see Exercise 19. For further details about constructions of generalized hashing ensembles, see Section 6.6.5.

An alternative presentation: The proofs of Propositions 6.3.6 and 6.3.7 actually establish that $\{g_{s,r} = f_s \circ h_r\}_{s \in \{0,1\}^*, r \leftarrow I(1^{|s|})}$ is a generalized pseudorandom function (as in Definition 3.6.12). Hence, the actual claim of these propositions (i.e., the security of the constructed MAC) can be derived from the fact that any generalized pseudorandom function yields a full-fledged MAC (see Exercise 7).

6.3.2 * More on Hash-and-Hide and state-based MACs

The basic idea underlying Construction 6.3.5 (as well as Proposition 6.3.7) is to combine a "weak tagging scheme" with an adequate "hiding scheme". Specifically, the "weak tagging scheme" should be secure against forgery *provided that the adversary does not have access to the scheme's outcome*, and the "hiding scheme" implements the latter provision in a setting in which the actual adversary does obtain the value of the MAC. In Construction 6.3.5 (and in Proposition 6.3.7), hiding was obtained by applying a pseudorandom function to the string that one wishes to hide. (Although this process is not 1-1, its result looks random and thus is hard to predict.)

One more natural "hiding scheme" (which can also be implemented using pseudorandom functions) is obtained by using certain private-key encryption schemes. For example, we may use Construction 5.3.9 (in which the plaintext x is encrypted/hidden by the pair $(y, x \oplus f_s(y))$, where y is uniformly selected), instead of hiding x by the value $f_s(x)$ (as above). Alternative implementations of this underlying idea are more popular, especially in the context of state-based MACs. We start by defining state-based MACs, and then show how to construct them based on the hash-and-hide (or rather tag-and-hide) paradigm.

6.3.2.1 The definition of state-based MACs

As in the case of steam-ciphers discussed in Section 5.3.1, we extend the mechanism of message-authentication schemes (MACs) by allowing the signing and verification processes to maintain and update a state. Formally, both the signing and the verification algorithms take an additional input and emit an additional output, corresponding to their state before and after the operation. The length of the state is not allowed to grow by too much during each application of the algorithm (see Item 3 below), or else efficiency of the entire "repeated signing" process can not be guaranteed. For sake of simplicity, we incorporate the key in the state of the corresponding algorithm. Thus, the initial state of each of the algorithms is set to equal its corresponding key. Furthermore, one may think of the intermediate states as of updated values of the corresponding key.

In the following definition, we follow similar conventions to those used in defining state-based ciphers (i.e., Definition 5.3.1). Specifically, for simplicity, we assume that the verification algorithm (i.e., V) is deterministic (otherwise the formulation would be more complex). Intuitively, the main part of the verification condition (i.e., Item 2) is that the (proper) iterative signing-verifying process always accepts. The additional requirement in Item 2 is that the state of the verification algorithm is updated correctly as long as it is fed with strings of length equal to the length of the valid document-signature pairs. The importance of this condition was discussed in Section 5.3.1 and is further discussed below.

Definition 6.3.8 (state-based MAC – the mechanism): *A* state-based message-authentication scheme *is a triple, (G,S,V), of probabilistic polynomial-time algorithms satisfying the following three conditions*

1. *On input 1^n, algorithm G outputs a pair of bit strings.*
2. *For every pair $(s^{(0)},v^{(0)})$ in the range of $G(1^n)$, and every sequence of $\alpha^{(i)}$'s, the following holds: if $(s^{(i)},\beta^{(i)}) \leftarrow S(s^{(i-1)},\alpha^{(i)})$ and $(v^{(i)},\gamma^{(i)}) \leftarrow V(v^{(i-1)},\alpha^{(i)},\beta^{(i)})$ for $i=1,2,...$, then $\gamma^{(i)}=1$ for every i. Furthermore, for every i and every $(\alpha,\beta) \in \{0,1\}^{|\alpha^{(i)}|} \times \{0,1\}^{|\beta^{(i)}|}$, it holds that $V(v^{(i-1)},\alpha,\beta)=(v^{(i)},\cdot)$.*
3. *There exists a polynomial p such that for every pair $(s^{(0)},v^{(0)})$ in the range of $G(1^n)$, and every sequence of $\alpha^{(i)}$'s and $s^{(i)}$'s as above, it holds that $|s^{(i)}| \leq |s^{(i-1)}| + |\alpha^{(i)}| \cdot p(n)$. Similarly for the $v^{(i)}$'s.*

That is, as in Definition 6.1.1, the signing-verification process operates properly provided that the corresponding algorithms get the corresponding keys (states). Note that in Definition 6.3.8 the keys are modified by the signing-verification process, and so correct verification requires holding the correctly-updated verification-key. We stress that the furthermore clause in Item 2 guarantees that the verification-key is correctly updated as long as the verification process is fed with strings of the correct lengths (but not necessarily with the correct document-signature pairs). This extra requirement implies that given the initial verification-key and the current document-signature pair as well as the lengths of all previous pairs (which may be actually incorporated in the current signature), one may correctly decide whether or not the current document-signature pair is valid. As in case of state-based ciphers (cf. Section 5.3.1), this fact is interesting for two reasons:

A theoretical reason: It implies that, without loss of generality (alas with possible loss in efficiency), the verification algorithm may be stateless. Furthermore, without loss of generality (alas with possible loss in efficiency), the state of the signing algorithm may consist of the initial signing-key and the lengths of the messages signed so far. (We assume here and below that the length of the signature is determined by the length of the message and the length of the signing-key.)

A practical reason: It allows to recover from the loss of some of the message-signature pairs. That is, assuming that all messages have the same length (which is typically the case in MAC applications), if the receiver knows (or is given) the total number of messages sent so far then it can verify the authenticity of the current message-signature pair, even if some of the previous message-signature pairs were lost.

We stress that Definition 6.3.8 refers to the signing of multiple messages (and is meaningless when considering the signing of a single message). However, Definition 6.3.8 (by itself) does not explain why one should sign the ith message using the updated signing-key $s^{(i-1)}$, rather than by reusing the initial signing-key $s^{(0)}$ (where all corresponding verifications are done by reusing the initial verification-key $v^{(0)}$). Indeed, the reason for updating these keys is provided by the following security definition that refers to the signing of multiple messages, and holds only in case the signing-keys in use are properly updated (in the multiple-message authentication process).

Definition 6.3.9 (security of state-based MACs):

- *A* chosen message attack on a state-based MAC, *(G,S,V), is an interactive process that is initiated with $(s^{(0)},v^{(0)}) \leftarrow G(1^n)$, and proceed as follows: In the i^{th} iteration, based on the information gathered so far, the attacker selects a string $\alpha^{(i)}$, and obtains $\beta^{(i)}$, where $(s^{(i)},\beta^{(i)}) \leftarrow S(s^{(i-1)},\alpha^{(i)})$.*

- *Such an attack is said to* succeeds *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) such that α is different from* *all signature-queries made during the attack, and* $V(v^{(i-1)}, \alpha, \beta) = (\cdot, 1)$ *holds for some intermediate state (verification-key)* $v^{(i-1)}$ (as above).[11]

- *A state-based MAC is* secure *if every probabilistic polynomial-time chosen message attack as above succeeds with at most negligible probability.*

Note that Definition 6.3.9 (only) differs from Definition 6.1.2 in the way that the signatures $\beta^{(i)}$'s are produced (i.e., using the updated signing-key $s^{(i-1)}$ rather than the initial signing-key $s^{(0)}$). Furthermore, Definition 6.3.9 guarantees nothing regarding a signing process in which the signature to the ith message is obtained by invoking $S(s^{(0)}, \cdot)$ (as in Definition 6.1.2).

6.3.2.2 State-based hash-and-hide MACs

We are now ready to present alternative implementations of the hash-and-hide paradigm. Recall that in Section 6.3.1.3, the document was hashed (by using an adequate hashing function) and the resulting hash-value was (authenticated and) hidden by applying a pseudorandom function to it. In the current subsection, hiding will be obtained in a more natural (and typically more efficient) way; that is, by XORing the hash-value with a new portion of a (pseudorandom) one-time pad. Indeed, the state is used in order to keep track of what part of the (one-time) pad was already used (and should not be used again). Furthermore, to obtain improved efficiency, we let the state encode information that allows fast generation of the next portion of the (pseudorandom) one-time pad. This is obtained using (on-line) pseudorandom generator (see Sections 3.3.3 and 5.3.1).

Recall that on-line pseudorandom generators are a special case of variable-output pseudorandom generators (see Section 3.3.3), in which a hidden state is maintained and updated so to allow generation of the next output bit in time polynomial in the length of the initial seed, regardless of the number of bits generated so far. Specifically, the next (hidden) state and output bit are produced by applying a (polynomial-time computable) function $g : \{0,1\}^n \to \{0,1\}^{n+1}$ to the current state (i.e., $(s', \sigma) \leftarrow g(s)$, where s is the current state, s' is the next state and σ is the next output bit). Analogously to Construction 5.3.3, the suggested state-based MAC will use an on-line pseudorandom generator in order to generate the required pseudorandom one-time pad, and the latter will be used to hide (and authenticate) the hash-value (obtained by hashing the original document).

Construction 6.3.10 (a state-based MAC): *Let* $g : \{0,1\}^* \to \{0,1\}^*$ *such that*

[11] In fact, one may strengthen the definition by using a weaker notion of success in which it is only required that $\alpha \neq \alpha^{(i)}$ (rather than requiring that $\alpha \notin \{\alpha^{(j)}\}_j$). That is, the attack is successful if, for some i, it outputs a pair (α, β) such that $\alpha \neq \alpha^{(i)}$ and $V(v^{(i-1)}, \alpha, \beta) = (\cdot, 1)$, where the $\alpha^{(j)}$'s and $v^{(j)}$'s are as above. The stronger definition provides "replay protection" (i.e., even if the adversary obtains a valid signature that authenticates α as the jth message it cannot produce a valid signature that authenticates α as the ith message, unless α was actually authenticated as the ith message).

$|g(s)| = |s|+1$, *for every* $s \in \{0,1\}^*$. *Let* $\{h_r : \{0,1\}^* \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^*}$ *be a family of functions having an efficient evaluation algorithm.*

key-generation and initial state: *Uniformly select* $s, r \in \{0,1\}^n$, *and output the key-pair* $((s,r),(s,r))$. *The initial state of each algorithm is set to* $(s,r,0,s)$.

(We maintain the initial key (s,r) *and a step-counter in order to allow recovery from loss of message-signature pairs.)*

signing message x with state (s,r,t,s'): *Let* $s_0 \stackrel{\rm def}{=} s'$. *For* $i = 1,...,m(n)$, *compute* $s_i\sigma_i = g(s_{i-1})$, *where* $|s_i| = n$ *and* $\sigma_i \in \{0,1\}$. *Output the signature* $h_r(x) \oplus \sigma_1 \cdots \sigma_{m(n)}$, *and set the new state to* $(s,r,t+m(n),s_{m(n)})$.

verification of the pair (x,y) with respect to the state (s,r,t,s'): *Compute* $\sigma_1 \cdots \sigma_{m(n)}$ *and* $s_{m(n)}$ *as in the signing process; that is, for* $i = 1,...,m(n)$, *compute* $s_i\sigma_i = g(s_{i-1})$, *where* $s_0 \stackrel{\rm def}{=} s'$. *Set the new state to* $(s,r,t+m(n),s_{m(n)})$, *and accept if and only if* $y = h_r(x) \oplus \sigma_1 \cdots \sigma_{m(n)}$.

When notified that some message-signature pairs may have been lost and that the current message-signature pair has index t', *one first recovers the correct current state, which as above will be denoted* s_0. *This is done by setting* $s_{-t'} \stackrel{\rm def}{=} s$ *and computing* $s_{i-t'}\sigma_{i-t'} = g(s_{i-t'-1})$, *for* $i = 1,...,t'$.

Note that both the signing and verification algorithms are deterministic, and that the state after authentication of t messages has length $3n + \log_2(t \cdot m(n)) < 4n$ (for $t < 2^n/m(n)$).

We now turn to analyze the security of Construction 6.3.10. The hashing property of the collection of h_r's should be slightly stronger than the one used in Section 6.3.1.3. Specifically, rather than a bound on the collision probability (i.e., the probability that $h_r(x) = h_r(y)$ for any relevant fixed x, y and a random r), we need a bound on the probability that $h_r(x) \oplus h_r(y)$ equals any fixed string (again, for any relevant fixed x, y and a random r). This property is commonly referred to by the name *Almost-Xor-Universal* (AXU). That is, $\{h_r : \{0,1\}^* \to \{0,1\}^{m(|r|)}\}_{r\in\{0,1\}^*}$ is called a (ℓ,ε)-AXU family if for every $n \in \mathbb{N}$, every $x \neq y$ such that $|x|, |y| \leq \ell(n)$, and every z, it holds that

$$\mathsf{Pr}[h_{U_n}(x) \oplus h_{U_n}(y) = z] \;\; \leq \;\; \varepsilon(n) \qquad (6.4)$$

References to constructions of such families are provided in Section 6.6.5.

Proposition 6.3.11 *Suppose that* g *is a pseudorandom generator, and that* $\{h_r\}$ *is a* (ℓ,ε)*-AXU family, for some super-polynomial* ℓ *and negligible* ε. *Then Construction 6.3.10 constitutes a secure state-based MAC. Furthermore, security holds even with respect to the stronger notion discussed in Footnote 11.*

Proof Sketch: By Exercise 21 of Chapter 3, if g is a pseudorandom generator then for every polynomial p the ensemble $\{G_n^p\}_{n\in\mathbb{N}}$ is pseudorandom, where G_n^p is defined by the following random process:

Uniformly select $s_0 \in \{0,1\}^n$;
For $i = 1$ to $p(n)$, let $s_i\sigma_i \leftarrow g(s_{i-1})$, where $\sigma_i \in \{0,1\}$ (and $s_i \in \{0,1\}^n$);
Output $\sigma_1\sigma_2 \cdots \sigma_{p(n)}$.

Recall that, in such a case, we said that g is a next-step function of an on-line pseudorandom generator.

As in previous cases, it suffices to establish the security of an ideal scheme in which the sequence (of $m(n)$-bit long blocks) produced by iterating the next-step function g is replaced by a truly random sequence (of $m(n)$-bit long blocks). In the ideal scheme, all that the adversary may obtain via a chosen message attack is a sequence of $m(n)$-bit long blocks, which is uniformly distributed among all such possible sequences. Note that each of the signatures obtained during the attack as well as the forged signature refers to a single block in this sequence (e.g., the ith obtained signature refers to the ith block). We consider two types of forgery attempts:

1. In case the adversary tries to forge a signature referring to an *unused* (during the attack) block, it may succeed with probability at most $2^{-m(n)}$, because we may think of this block as being chosen after the adversary makes its forgery attempt. Note that $2^{-m(n)}$ is negligible, because $\varepsilon(n) \geq 2^{-m(n)}$ must hold (since $2^{-m(n)}$ lower-bounds the collision probability).

2. The more interesting case is when the adversary tries to forge a signature referring to a block, say the ith one, that was used (to answer the ith query) during the attack. Denote the jth query by $\alpha^{(j)}$, the (random) jth block by $b^{(j)}$, and the forged document by α. Then, at the time of outputting the forgery attempt (α, β), the adversary only knows the sequence of $b^{(j)} \oplus h_r(\alpha^{(j)})$'s, which yields no information on r. Note that the adversary succeeds if and only if $b^{(i)} \oplus h_r(\alpha) = \beta$, where $\beta^{(i)} \stackrel{\text{def}}{=} b^{(i)} \oplus h_r(\alpha^{(i)})$ is known to it. Thus, the adversary succeeds if and only if $h_r(\alpha^{(i)}) \oplus h_r(\alpha) = \beta^{(i)} \oplus \beta$, where $\alpha^{(i)}, \beta^{(i)}, \alpha, \beta$ are fixed and r is uniformly distributed. Hence, by the AXU property, the probability that the adversary succeeds is at most $\varepsilon(n)$.

The security of the real scheme follows (or else one could have distinguished the sequence produced by iterating the next-step function g from a truly random sequence). ■

Construction 6.3.10 versus the constructions of Section 6.3.1.3. Recall that all these schemes are based on the hash-and-hide paradigm. The difference between the schemes is that in Section 6.3.1.3 a pseudorandom function is applied to the hash-value (i.e., the signature to x is $f_s(h_r(x))$), whereas in Construction 6.3.10 the hash-value is XORed with a pseudorandom value (i.e., we may view the signature as consisting of $(c, h_r(x) \oplus f_s(c))$, where c is a counter value and $f_s(c)$ is the cth block produced by iterating the next-step function g starting with the initial seed s). We note two advantages of the state-based MAC over the MACs presented in Section 6.3.1.3: First, applying an on-line pseudorandom generator is likely to be more efficient than applying a pseudorandom function. Second, a counter allows to securely authenticate more messages than can be securely authenticated by applying a pseudorandom function to the hashed value. Specifically, the use of an a m-bit long counter allows to securely authenticate 2^m messages, whereas using an m-bit long hash-value suffers from the "birthday effect" (i.e., collisions are likely to occur when $\sqrt{2^m}$ messages are authenticated). Indeed, these advantages are relevant only in applications in which using state-based MACs is possible, and are most advantageous in applications where verification is performed in the same order as signing (e.g., in FIFO communication).

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 crucial 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 crucial 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 the "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 8.

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 define a length-restricted version of this notion (analogous to Definition 6.2.1), then present a simple construction of the latter, and finally we 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 only care 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*

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

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* $\Pr[V(v, \alpha, S(s, \alpha))=1] = 1$.

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 $|\alpha| = \ell(n)$.*

Note that even the existence of secure 1-restricted one-time signature schemes implies the existence of one-way functions: see Exercise 11.

6.4.1.2 Constructing 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 alleged 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 random image (corresponding to a bit location on which the two $\ell(n)$-bit long messages differ). 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 uniformly and independently a position p in $\{1, ..., \ell(n)\}$, a bit b, and a sequence of ($2\ell(n)$ many) n-bit long strings $s_1^0, s_1^1, ..., s_{\ell(n)}^0, s_{\ell(n)}^1$. (Actually, s_p^b 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 $v_p^{1-b} = f(s_p^{1-b})$, and sets $v_p^b = y$ 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 b, 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 $\sigma_p = b$. If this event occurs, A' obtains a preimage of y under f, since the validity of the signature implies that $f(\tau_p) = v_p^{\sigma_p} = v_p^b = y$. Observe that conditioned on the value of v and the internal coin tosses of A, the value b 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 9.) In this case, on input y (supposedly in the range of f), algorithm A' selects p, b and the s_i^j's, and forms the v_i^j's and v exactly as in the warm-up above.[12] Recall that if $y = f(x)$, for a uniformly distributed $x \in \{0,1\}^n$, then for each possible choice of p and b, 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 $v_p^b = y$, algorithm A' holds a random preimage (of v_i^j) 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 this query:

1. If $\sigma_p = b$ then A' can not supply the desired signature since it lacks a preimage of $s_p^b = y$ 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 b).

   (That is, conditioned on the value of v and the internal coin tosses of A, this case occurs with probability $\frac{1}{2}$.)[13]

2. If $\sigma_p = 1 - b$ 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-b}. 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 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., $\sigma_p = 1-b$), algorithm A outputs a forged message–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).[14] Let $\alpha' = \sigma'_1 \cdots \sigma'_{\ell(n)}$ and $\beta' = s'_1 \cdots s'_{\ell(n)}$, where (α', β') is the forged message–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 $v_p^{1-\sigma_p} = v_p^b = y$).

[12] That is, first A' selects p uniformly in $\{1, ..., \ell(n)\}$, b 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 $v_p^{1-b} = f(s_p^{1-b})$, and sets $v_p^b = y$ and $v = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$.

[13] 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, b is uniformly distributed in $\{0,1\}$ (and so $\sigma_p = b$ happens with probability exactly $\frac{1}{2}$).

[14] Alternatively, recall that, without loss of generality, we may assume that A always makes a single query; see Exercise 9.

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 construct (in that proof) to attack (G,S,V). In general, the adversary A constructed in the proof of Proposition 6.2.7 makes a single 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 12), 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') only 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, we later avoid (in Section 6.4.3) the use of collision-free hashing. Instead, we use "universal one-way hashing functions" (to be defined), and present a variant 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* (general) *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 each new document. Of course, these fresh instances should be authenticated by the original instance (corresponding to the verification-key that is publically known), but such an authentication refers to a string selected by the legitimate signer rather than by the adversary.

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 alleged 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 launched 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 obtain $(s', v') \leftarrow G'(1^n)$. Next, invoke S to obtain $\beta_1 \leftarrow S_s(v')$, and S' to obtain $\beta_2 \leftarrow S'_{s'}(\alpha)$. 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 alleged 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 one-time verification-keys included in the signatures provided by S''_s and the one-time verification-key included 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'). Furthermore, 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' makes at most one 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'').)[15]

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 authenticate this fresh instance relative to the single verification-key that is public. A straightforward implementation of this

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

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 (of the one-time signature scheme), 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 6.2 (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).

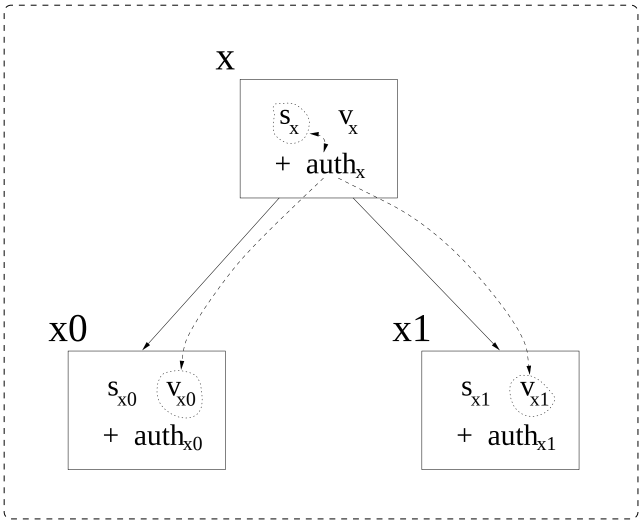

Figure 6.2: A node labeled x authenticates its children, labeled $x0$ and $x1$, respectively. The authentication is via a one-time signature of the text $v_{x0}v_{x1}$ using signing-key s_x.

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 all previous invocations of the signing process.[16] We refer to such a scheme as *memory dependent*.

Definition 6.4.13 (memory-dependent signature schemes):

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

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 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.*

The definition of memory-dependent signature schemes (i.e., Definition 6.4.13) is related to the definition of state-based MACs (i.e., Definition 6.3.9). However, there are two differences between the two definitions: First, Definition 6.4.13 refers to (public-key) signature schemes, whereas Definition 6.3.9 refers to MACs. Second, in Definition 6.4.13 only the signing algorithm is state-based (or memory-dependent), whereas in Definition 6.3.9 also the verification algorithm is state-based. The latter difference reflects the difference in the applications envisioned for both types of schemes. (Typically, MACs are indented for communication between a predetermined set of "mutually synchronized" parties, whereas signature schemes are intended for production of signatures that may be universally verifier at any time.)

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 record 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 retrieve 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 retrieve 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* $S_{s_{\sigma_1 \cdots \sigma_{i-1}}}(v_{\sigma_1 \cdots \sigma_{i-1}0}\, v_{\sigma_1 \cdots \sigma_{i-1}1})$*. (The ability to retrieve this signature from memory for repeated use is the most important place in which we rely on the memory-dependence of our signature scheme.)*[17]

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 labeled $\sigma_1 \cdots \sigma_{i-1}$ 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 alleged 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 $v_{\sigma_1 \cdots \sigma_i}$.)

[17] This allows the signing process S'_s to use each (one-time) signing-key s_x for producing a single S_{s_x}-signature. In contrast, the use of a counter for determining a new leaf can be easily avoided, by selecting a leaf at random.

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 believed 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 believed 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 (when signing a message using the leaf associated with $\sigma_1 \cdots \sigma_n$). This excess is merely an artifact of the need to use $s_{\sigma_1 \cdots \sigma_i}$ only once 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

$$(\sigma_1 \cdots \sigma_n, \text{auth}_\lambda, \text{auth}_{\sigma_1}, ..., \text{auth}_{\sigma_1 \cdots \sigma_{n-1}}, S_{s_{\sigma_1 \cdots \sigma_n}}(\alpha)) \tag{6.5}$$

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

$$\text{auth}_x = (v_{x0}\,,\, v_{x1}\,,\, S_{s_x}(v_{x0}\, v_{x1})) \tag{6.6}$$

(See Figure 6.2.) 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 $(\text{auth}_\lambda, \text{auth}_{\sigma_1}, ..., \text{auth}_{\sigma_1 \cdots \sigma_{i-1}})$ 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).[18] 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 $\text{auth}_{\sigma_1\cdots\sigma_{i-1}}$ (produced by the forge), contains a signature relative to (G, S, V) using the signing-key $s_{\sigma_1\cdots\sigma_{i-1}}$. 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) generated during the attack of A'. This instance will be attacked by A. Note that since $2n+1$ instances of (G, S, V) are referred to in each signature relative to (G', S', V'), the

[18] 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 during the entire attack of A'. 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 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 exception refers 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 record 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 retrieve 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 retrieve 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 identified 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 (and retrieves this signature from memory if it ever needs it again).

  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,[19] we need 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:

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

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 identified 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

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

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 $v'_{i-1,\sigma'_i} = v_{\sigma'_1 \cdots \sigma'_i}$, 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 (possibly) only 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. ■

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 that 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 exactly 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. Likewise, 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).[20] 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. Consider 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.*[21]

[20] 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 $x10^{n+2-i}$).

[21] In case we use ordinary pseudorandom functions, rather than generalized ones, we select

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 forgery (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.)[22]

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.*[23]

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.*[24]

4. *Finally, the algorithm invokes $S_{s_{\sigma_1 \cdots \sigma_n}}$ and outputs*[25]

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

verification algorithm V': *On input a verification-key v, a document α, and an alleged 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:

---

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 dropping the last 3 bits of the function value.

[22] 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.

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

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

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

- $V_v(v_{0,0}v_{0,1},\beta_0) = 1$.
- *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* (general) *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 truly 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 stored 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 truly 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 truly 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 succeeds only with negligible probability when attacking the ideal scheme. We construct a distinguisher D that on input 1^n and oracle access to $f : \{0,1\}^* \to \{0,1\}^n$ behaves 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 alleged signature to a new document, machine M evaluates whether or not 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 truly 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 11), 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 choice of parameters in Constructions 6.4.14 and 6.4.16 (i.e., $d \equiv 2$ and $e(n) = n$) is probably the simplest one as well as the most efficient one (from a generic perspective).

  Natural complexity measures for a signature scheme include the length of signatures and the signing and verification times. In a generalized construction, the length of the signatures is linear in $d(n) \cdot e(n)$, and the number of applications of the underlying one-time signature scheme (per each general signature) is linear in $e(n)$, where in internal nodes the one-time signature scheme is applied to string of length linear in $d(n)$. Assuming that the complexity of one-time signatures is linear in the document length, all complexity measures are linear in $d(n) \cdot e(n)$, and so $d \equiv 2$ is the best generic choice. However, the above assumption may be wrong when some specific one-time signatures are used. For example, the complexity of producing a signature to an ℓ-bit long string in a one-time signature scheme may be of the form $p(n) + p'(n) \cdot \ell$, where $p'(n) \ll p(n)$. In such (special) cases, one may prefer to use larger $d : \mathbb{N} \rightarrow \mathbb{N}$ (see Section 6.6.5).

- 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^{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.[26] 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 (i.e., of size merely greater than the total number of documents that should be ever signed).

More generally, we wish to stress that each of the following ingredients of the above constructions, is useful in a variety of related and unrelated settings. We refer specifically to the *refreshing paradigm*, the *authentication tree construction*, and the notion (and constructions) of *one-time signatures*. For example:

- It is common practice to authenticate messages sent during a "communication session" via a session-key that is typically authenticated by a master-key. One of the reasons for this practice is the prevention of a chosen message attack on the (more valuable) master-key. (Other reasons include allowing the use of a faster alas less secure authentication scheme for the actual communication, introducing independence between sessions, etc.)

- Observe the analogy between the tree-hashing (of Construction 6.2.13) and the authentication tree (of Construction 6.4.14). Despite the many differences, in both cases, the value of internal nodes essentially determines the values that may be claimed for their children.

- Recall the application of one-time signatures in the construction of CCA-secure public-key encryption schemes (cf. proof of Theorem 5.4.30).

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 11). 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

[26] 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 (e.g., 1/20).

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, 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 of the function) 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):[27] *For some polynomial p, all sufficiently large n's and every s in the range of $I(1^n)$ it holds that $n \leq p(|s|)$. Furthermore, n can be computed in polynomial-time from 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*

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

algorithm A, every polynomial p and all sufficiently large n's

$$\mathsf{Pr}\left[\begin{array}{c} h_{I(1^n)}(A(I(1^n),U_{q(n)})) = h_{I(1^n)}(A_0(U_{q(n)})) \\ \text{and} \quad A(I(1^n),U_{q(n)}) \neq A_0(U_{q(n)}) \end{array}\right] < \frac{1}{p(n)} \tag{6.7}$$

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 (initial) adversary generates a preimage $x = A_0(r)$; next, a function h is selected; and, finally, the (residual) 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. (6.7) 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 15). Finally, note that any collision-free hashing is universally one-way hashing, but the converse is false (see Exercise 16). 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).

6.4.3.2 Constructions

We construct UOWHF collections in several steps, starting with a related but restricted notion, and relaxing the restriction gradually (until we reach the unrestricted notion of UOWHF collections). The abovementioned restriction refers to 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)-UOWHFs): *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 $|s| = n$.*[28]
2. *There exists a polynomial-time algorithm that given s and $x \in \{0,1\}^{d(|s|)}$, returns $h_s(x)$.*

[28] Here we chose to make a more stringent condition, requiring that $|s| = n$ rather than $n \leq \text{poly}(|s|)$. 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 $l(n) \leq \text{poly}(n)$ and $n \leq \text{poly}(l(n))$.

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. (6.7) 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 $d(n) \leq r(n)$ 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. Furthermore, the construction of such a minimal $(d,d-1)$-UOWHF (undertaken in the first step) is the most interesting step to be taken on our entire way towards the construction of full-fledged UOWHF.

Step I: constructing $(d,d-1)$-UOWHFs. We show how to construct length-restricted UOWHFs 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 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 $h'_s \stackrel{\rm def}{=} h'_{s'}$ 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 $p(n) \leq m$. Note that d is monotonically non-decreasing, and that for 1-1 p's the corresponding d is onto (i.e., $d(p(n)) = n$ for every n).

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 hashing 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 the additional properties required in the next proposition (see Exercise 20).

Proposition 6.4.21 *Suppose that f is a one-way permutation, and that* $\log_2|S_n^{n-1}| = 2n$. *Furthermore, suppose that S_n^{n-1} satisfies the following two conditions*

C1 *All but a negligible fraction of the functions in S_n^{n-1} are 2-to-1.*

C2 *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, because the collision is due to h_s, which may be selected such that $h_s(y) = h_s(f(x'))$ for any given y and x'. We stress that typically there are only two preimages of $h'_s(x')$ under h'_s, one being x' itself (which is given to the collision-finder) and the other being $f^{-1}(y)$ such that $h_s(y) = h'_s(x')$. Thus, if we wish to invert f on a random image y, then we may invoke a collision-finder, which first outputs some x', supply it with a random s satisfying $h_s(y) = h'_s(x')$, and hope that it forms a collision (i.e., finds a different preimage x satisfying $h'_s(x) = h'_s(x')$). Indeed, the different preimage must be $f^{-1}(y)$, which means that whenever the collision-finder succeed we also succeed (i.e., invert f on 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, which 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 Condition C2, 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 Condition C1, 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'' such that $x'' \neq x'$ and $h'_s(x'') = h'_s(x')$ then $f(x'') = y$.

Proving the Claim: By the definitions of h'_s and A (i.e., its Step (2)), we have $h'_s(x) = h_s(f(x)) = h_s(y) = h_s(y') = h_s(f(x')) = h'_s(x')$, which equals $h'_s(x'')$ by one of the claim's hypotheses. Thus, x', x'' and x are all preimages of $h'_s(x) = h_s(y)$ under h'_s, but they are not necessarily distinct. By other two hypotheses $x' \neq x''$ and $h'_s = h_s \circ f$ is 2-to-1 (since h_s is 2-to-1 and f is 1-to-1). Thus, $x \in \{x', x''\}$. Using the last of the claim's hypotheses (i.e., $y = f(x) \neq f(x')$) and the hypothesis that f is 1-1, it follows that $x \neq x'$, which in turn implies that $x = x''$ 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 (accounting for the negligible probability that h_s is not 2-to-1). The proposition follows. □

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 construct a $(d', d'/2)$-UOWHF. That is, we construct length-restricted UOWHFs that shrink their input by a factor of 2. The construction is obtained by composing a sequence of (different) $(d, d-1)$-UOWHFs. 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*

$$h'_{s_1,...,s_{\lfloor d(n)/2 \rfloor}}(x) \stackrel{\rm def}{=} h_{s_{\lfloor d(n)/2 \rfloor}}(\cdots h_{s_2}(h_{s_1}(x)) \cdots)$$

That is, we let $x_0 \stackrel{\rm def}{=} x$, and $x_i \leftarrow h_{s_i}(x_{i-1})$, for $i = 1, ..., \lfloor d(n)/2 \rfloor$. (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 such that $m \geq m' \stackrel{\rm def}{=} \sum_{i=1}^{\lfloor d(n)/2 \rfloor} d^{-1}(d(n) + 1 - i)$, 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 $h'_{s_0,s_1,...,s_{\lfloor d(n)/2 \rfloor}} \stackrel{\rm def}{=} h'_{s_1,...,s_{\lfloor d(n)/2 \rfloor}}$.

That is, $m = |\overline{s}|$ and $h'_{\overline{s}} : \{0,1\}^{d(n)} \to \{0,1\}^{\lfloor d(n)/2 \rfloor}$, 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 such 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 $p(d(n)) \geq n$. (E.g., for every fixed $\varepsilon, \varepsilon' > 0$, the function $d(n) = \varepsilon' n^{\varepsilon}$ is sufficiently-growing.)

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 $x_0 \stackrel{\rm def}{=} x$, 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 following details are quite tedious, and merely provide an implementation of the above idea.

As stated, 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 such 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 start with the description of A_0; that is, the algorithm that generates preimages of $\{h_s\}$. Intuitively, A_0 selects a random j, uses A'_0 to obtain a preimage x_0 of $\{h'_{\overline{s}}\}$, generates random $s_0, ..., s_{j-1}$, and outputs a preimage x_{j-1} of $\{h_{s_j}\}$, computed by $x_i = h_{s_i}(x_{i-1})$ for $i = 1, ..., j-1$. (Algorithm A will be given x_{j-1} and a random $h_{s_{j-1}}$ and will try to form a collision with x_{j-1} under $h_{s_{j-1}}$.) Specifically, on input $r \in \{0,1\}^{p(n)}$, algorithm A_0 proceeds as follows, where $q(n) \stackrel{\rm def}{=} d^{-1}(2d(n))$.

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. Algorithm A will be given x_{j-1} as produced above (along with (or actually only) the coins used in its generation) as well as a random $h_{s_{j-1}}$ and will try to form a collision with x_{j-1} under $h_{s_{j-1}}$. 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 preimages 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 $\varepsilon(n) \stackrel{\text{def}}{=} \frac{\varepsilon'(m)}{m^2 \cdot d'(n')} = \frac{\varepsilon'(\text{poly}(n))}{\text{poly}(n)}$, with respect to preimages produced by A_0. Thus, if ε' is non-negligible then so is ε, and the proposition follows. □

Step III: Constructing (length-unrestricted) quasi-UOWHFs that shrink their input by a factor of two. The third step on our way consists of using any $(d, d/2)$-UOWHF in order to construct "quasi 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 resulting construct does not fit Definition 6.4.19, since the function's output length depends on the function's input length, but the function 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' (e.g., $d'(n) = n^2$), whereas in Construction 6.4.22 the function d' is fixed and satisfies $d'(n) \ll n$. The construction itself amounts to parsing the input into blocks and applying the same $(d, d/2)$-UOWHF to each block.

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 $s \in \{0,1\}^n$ and every $x \in \{0,1\}^*$, we define*

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

where $x = x_1 \cdots x_t$, $0 \le |x_t| < d(n)$ 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 such 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 preimages produced by a deterministic polynomial-time algorithm A'_0, we construct algorithms A_0 and A such that A forms designated collisions under $\{h_s\}$ with respect to preimages 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-tape used by algorithm A_0 has the form (r', i), and A_0 outputs the $i^{\rm th}$ block in the parsing of the string $A'_0(r')$. 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 $i^{\rm th}$ block in the standard parsing of x'.

Note that whenever we have a collision under h'_s (i.e., a pair $x \neq x'$ such that $h'_s(x) \neq h'_s(x')$), we obtain at least one collision under the corresponding h_s (i.e., for some i, the ith blocks of $x \neq x'$ differ, and yet both are mapped by h_s to the same image). Thus, 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 upper-bounds the length of the output of A', and so $1/t(n)$ is a lower bound on the probability that the colliding strings differ in a certain uniformly selected block). The proposition follows. $\square$

Step IV: Full-fledged UOWHFs. The last step on our way consists of using any quasi-UOWHFs as constructed (in Step III) above to obtain full-fledged

UOWHFs. That is, we use quasi-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). The construction is obtained by composing a sequence of (different) quasi-UOWHFs; that is, the following construction is analogous to Construction 6.4.22.

Construction 6.4.26 (a UOWHF): *Let* $\{h_s : \{0,1\}^* \to \{0,1\}^*\}_{s\in\{0,1\}^*}$, *so that* $|h_s(x)| \le |x|/2$, *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*

$$h'_{s_1,...,s_n}(x) \stackrel{\rm def}{=} h_{s_t}(\cdots h_{s_2}(h_{s_1}(x))\cdots)$$

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* $h'_{s_0,s_1,...,s_n} \stackrel{\rm def}{=} h'_{s_1,...,s_n}$.

Note that $h'_{s_0,s_1,...,s_n} : \{0,1\}^* \to \{0,1\}^n$, and that $|s_0, s_1, ..., s_n| = m < (n+1)^2$.

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 constitutes a collection of UOWHFs.*

The proof of Proposition 6.4.27 is omitted because it is almost identical to the proof of Proposition 6.4.23.

Conclusion: Combining the above four steps, we obtain a construction of (full-fledged) UOWHFs (based on any one-way permutation). That is, 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*. Thus, 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).[29] Such a construction is actually known, and so the following result is known to hold (but is not proven here):

[29] 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.

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 15.

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 value. 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 $\ell, \ell' : \mathbb{N} \to \mathbb{N}$ such that $\ell(n) = \ell'(n) + n$. Let (G, S, V) be an ℓ-restricted signature scheme as in Definition 6.2.1, and $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell'(|r|)}\}_{r\in\{0,1\}^*}$ be a collection of functions with an indexing algorithm I (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 to obtain $\beta_1 \leftarrow I(1^n)$.*
2. *Algorithm S' invokes S to produce $\beta_2 \leftarrow S_s(\beta_1, h_{\beta_1}(\alpha))$.*

Algorithm S' outputs the signature (β_1, β_2).

verification with V': *On input a verifying-key $v \in G'_2(1^n)$, a document $\alpha \in \{0,1\}^*$, and a alleged signature (β_1, β_2), algorithm V' invokes V, and outputs $V_v((\beta_1, h_{\beta_1}(\alpha)), \beta_2)$.*

Recall that secure ℓ-restricted one-time signature schemes exist for any polynomial ℓ, provided that one-way function exist. Thus, the fact that Construction 6.4.30 requires $\ell(n) > n$ is not a problem. In applying Construction 6.4.30, one should first choose a family of UOWHFs $\{h_r : \{0,1\}^* \to \{0,1\}^{\ell'(|r|)}\}_{r\in\{0,1\}^*}$, then determine $\ell(n) = \ell'(n)+n$, and use a corresponding secure ℓ-restricted one-time signature scheme.

Let us pause to compare Construction 6.2.6 with Construction 6.4.30. Recall that in Construction 6.2.6 the function description $\beta_1 \leftarrow I(1^n)$ is produced (and fixed as part of both keys) by the key-generation algorithm. Thus, the function description β_1 is trivially authenticated (i.e., by merely being part of the verification-key). Consequently, in Construction 6.2.6, the S'-signature (of α) equals $S_s(h_{\beta_1}(\alpha))$. In contrast, in Construction 6.4.30 a fresh new (function description) β_1 is selected per each signature, and thus β_1 needs to be authenticated. Hence, the S'-signature equals the pair $(\beta_1, S_s(\beta_1, h_{\beta_1}(\alpha)))$. Since we want to be able to use (length-restricted) *one-time* signatures, we let the signing algorithm authenticate both β_1 and $h_{\beta_1}(\alpha)$ via a single signature. (Alternatively, we could have used two instances of the signature scheme (G, S, V), one for signing the function description β_1, and the other for signing the hash value $h_{\beta_1}(\alpha)$.)

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 a collection of UOWHFs. Then* (G', S', V')*, as defined in Construction 6.4.30, 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 forgery 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 (i.e., the number of queries made by the forger constructed for (G, S, V) equals the number of queries made by the forger assumed (towards the contradiction) for (G', S', V')). 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). Specifically, to answer query q, algorithm A generates $a_1 \leftarrow I(1^n)$, forwards $(a_1, h_{a_1}(q))$ to its own oracle (i.e., S_s), and answers with (a_1, a_2), where $a_2 = S_s(a_1, h_{a_1}(q))$. (We stress that A issues a single S_s-query per each S'_s-query made by A'.) 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). That is, if A' outputs the document-signature pair (α, β), where $\beta = (\beta_1, \beta_2)$, then A will output the document-signature pair (α_2, β_2), where $\alpha_2 \stackrel{\rm def}{=} (\beta_1, h_{\beta_1}(\alpha))$.

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 $(\alpha^{(i)}, \beta^{(i)})$ denote the $i^{\rm th}$ query and answer pair made by A', and (α,β) be the forged document-signature pair that A' outputs (in case of success), where $\beta^{(i)} = (\beta_1^{(i)}, \beta_2^{(i)})$ and $\beta = (\beta_1, \beta_2)$. We consider the following two cases regarding the forging event:

Case 1: $(\beta_1, h_{\beta_1}(\alpha)) \neq (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$ for all i's. (That is, the S_s-signed value in the forged signature (i.e., $(\beta_1, h_{\beta_1}(\alpha))$) is different from all queries made to S_s.) In this case, the document-signature pair $((\beta_1, h_{\beta_1}(\alpha)), \beta_2)$ constitutes a success in existential forgery relative to the ℓ-restricted scheme (G, S, V).

Case 2: $(\beta_1, h_{\beta_1}(\alpha)) = (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$ 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, $\beta_1 = \beta_1^{(i)}$ and $h_{\beta_1}(\alpha) = h_{\beta_1^{(i)}}(\alpha^{(i)})$, although $\alpha \neq \alpha^{(i)}$. In this case, the pair $(\alpha, \alpha^{(i)})$ forms a designated collision under $h_{\beta_1^{(i)}}$ (and we do not obtain success in existential forgery relative to the ℓ-restricted scheme). We stress that A' selects $\alpha^{(i)}$ before it is given the description of the function $h_{\beta_1^{(i)}}$, and thus its ability to later produce $\alpha \neq \alpha^{(i)}$ such that $h_{\beta_1}(\alpha) = h_{\beta_1^{(i)}}(\alpha^{(i)})$ yields a violation of the UOWHF property.

Thus, if Case 1 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 Case 2 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 Case 2 follow.

We start with a sketch of the construction of an algorithm that attempts to form designated collisions under a randomly selected hash function. Loosely speaking, we construct an algorithm B' that tries to form designated collisions by emulating the attack of A' on an random instance of (G', S', V') that B' selects by itself. Thus, B' can easily answer any signing-query referred to it by A', but in one of these queries (the index of which B selects at random) algorithm B' will use a hash function given to it by the outside (rather than generating such a function at random by itself). In case A' forges a signature while using this specific function-value pair (as in Case 2), algorithm B' obtains and outputs a designated collision.

We now turn to the actual construction of algorithm B' (which attempts to form designated collisions under a randomly selected hash function). Recall that such an algorithm operates in three stages (see discussion preceding Definition 6.7): 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. Indeed, on input 1^n, algorithm B' proceeds in three stages:

Stage 1: Algorithm B' selects uniformly $i \in \{1, ..., t(n)\}$, where $t(n)$ bounds the running-time of $A'(G'_1(1^n))$ (and thus the number of queries it makes).

Next B' selects $(s,v) \leftarrow G'(1^n)$, and emulate the attack of $A'(v)$ on S'_s, while answering the queries of S'_s as follows. All queries except the i^{th} one are emulated in the straightforward manner (i.e., by executing the program of S'_s as stated). That is, for $j \neq i$, the j^{th} query, denoted $\alpha^{(j)}$, is answered by producing $\beta_1^{(j)} \leftarrow I(1^n)$, computing $\beta_2^{(j)} \leftarrow S_s(\beta_1^{(j)}, h_{\beta_1^{(j)}}(\alpha^{(j)}))$ (using the knowledge of s), and answering with the pair $(\beta_1^{(j)}, \beta_2^{(j)})$. The i^{th} query of A', denoted $\alpha^{(i)}$, will be used as the *designated preimage*. Once $\alpha^{(i)}$ is issued (by A'), algorithm B' completes its first stage (without answering this query), and the rest of the emulation of A' will be conducted by the third stage of B'.

Stage 2: At this point (i.e., after B' has selected the designated preimage $\alpha^{(i)}$), B' obtains a description of a random hashing function h_r (thus completing its second operation stage). That is, this stage consists of B' being given $r \leftarrow I(1^n)$.

Stage 3: Next, algorithm B' answers the i^{th} query (i.e., $\alpha^{(i)}$) by applying S_s to the pair $(r, h_r(\alpha^{(i)}))$. Subsequent queries are emulated in the straightforward manner (as explained above). When A' halts, B' checks whether A' has output a valid document-signature pair (α, β) as in Case 2 (i.e., $h_r(\alpha) = h_r(\alpha^{(j)})$ for some j), and whether the collision formed is indeed on the i^{th} query (i.e., $h_r(\alpha) = h_r(\alpha^{(i)})$). When this happens, B' outputs α, and doing so it succeeded in forming a designated collision (with $\alpha^{(i)}$ under h_r).

Now, if Case 2 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. ■

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 15), 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.*

Comment: the hash-and-sign paradigm, revisited. We wish to highlight the revised version of the hash-and-sign paradigm as underlying Construction 6.4.30. Similar to the original instantiation of the hash-and-sign paradigm (i.e., Construction 6.2.6), Construction 6.4.30 is useful in practice. We warn that using the latter construction requires verifying that (G, S, V) is a secure ℓ-restricted signature scheme and that $\{h_r\}$ is a UOWHF (rather than collision-free). The advantage of Construction 6.4.30 over Construction 6.2.6 is that the former relies on a seemingly weaker construct; that is, hardness of forming designated collisions (as in UOWHF) is a seemingly weaker condition than hardness of forming any collision (as in collision-free hashing). On the other hand, Construction 6.2.6 is simpler and more efficient (e.g., one need not generate a new hashing function per each signature).

6.5 * Additional Properties

We briefly discuss several properties of interest that some signature schemes enjoy. We first discuss properties that seem unrelated to the original purpose of signature schemes, but are useful towards utilizing signature scheme as a building block towards constructing other primitives (e.g., see Section 5.4.4.4). These (related) properties are having *unique valid signatures* and being *super-secure*, where the latter term indicates the infeasibility of finding a different signature even to a document for which a signature was obtained by the attack. We next turn to properties that offer some advantages in the originally-intended applications of signature schemes. Specifically, we consider properties that allow saving real time in some settings (see Sections 6.5.3 and 6.5.4), and a property supporting legitimate revoking of forged signatures (see Section 6.5.5).

6.5.1 Unique signatures

Loosely speaking, we say that a signature scheme (G, S, V) (either a private-key or a public-key one) has unique signatures if for every possible verification-key v and every document α there is a unique β such that $V_v(\alpha, \beta) = 1$.

Note that this property is related, but not equivalent, to the question of whether or not the signing algorithm is deterministic (which is considered in Exercise 1). Indeed, if the signing algorithm is deterministic then, for every key pair (s, v) and document α, the result of applying S_s to α is unique (and indeed $V_v(\alpha, S_s(\alpha)) = 1$). Still, this does *not* mean that there is no other β (which is never produced by applying S_s to α) such that $V_v(\alpha, \beta) = 1$. On the other hand, the unique signature property may hold even in case the signing algorithm is randomized, but indeed in this case the randomization can be eliminated from the latter (e.g., by replacing it with a fixed sequence in case the signing algorithm always succeeds, or incorporating the coins in the signing-key (and possibly using a pseudorandom function) otherwise).

Can secure signature schemes have unique signatures? The answer is definitely affirmative, and in fact we have seen several such schemes in the previous sections. Specifically, all private-key signature schemes presented in Section 6.3 have unique signatures. Furthermore, every secure private-key signature scheme can be transformed into one having unique signatures (e.g., by combining deterministic signing as in Exercise 1 with canonical verification as in Exercise 2). Turning to public-key signature schemes, we observe that if the one-way function f used in Construction 6.4.4 is 1-1, then the resulting secure length-restricted one-time (public-key) signature scheme has unique signatures (because each f-image has a unique preimage). In addition, Construction 6.2.6 (i.e., the basic hash-and-sign) preserves the unique signature property. Let use summarize all these observations.

Theorem 6.5.1 (secure schemes with unique signatures):

1. *Assuming the existence of one-way functions, there exist secure message authentication schemes having the unique signature property.*
2. *Assuming the existence of 1-1 one-way functions, there exist secure* length-restricted one-time (*public-key*) *signature schemes having the unique signature property.*
3. *Assuming the existence of 1-1 one-way functions and collision-free hashing collections, there exist secure* one-time (*public-key*) *signature schemes having the unique signature property.*

Still, this leaves open the question of whether or not there exist secure (full-fledged) signature schemes having the unique signature property.

6.5.2 Super-secure signature schemes

In case the signature scheme does not posses the unique signature property, it makes sense to ask whether given a message-signature pair it is feasible to produce a *different signature to the same message*. More generally, we may ask whether it is feasible for a chosen message attack to produce a different signature to any of the messages to which it has obtained signatures. Such ability may be of concern in some applications (but, indeed, not in the most natural applications). Combining the new concern with the standard notion of security, we derive the following notion, which we call super-security. A signature scheme is called super-secure if it is infeasible for a chosen message attack to produce a valid message-signature pair that is different from all query-answer pairs obtained during the attack, regardless of whether or not the message used in the new pair equals one of the previous queries. (Recall that ordinary security only requires the infeasibility of producing a valid message-signature pair such that the message part is different from all queries made in the attack.)

Do super-secure signature schemes exist? Indeed, every secure signature scheme that has unique signatures is super-secure, but the question is whether super-security may hold for a signature scheme that does not posses the unique signature property. We answer this question affirmatively.

Theorem 6.5.2 (super-secure signature schemes): *Assuming the existence of one-way functions, there exist super-secure (public-key) signature schemes.*

In other words, super-secure signature schemes exist if and only if secure signature schemes exist. We comment that the signature scheme constructed in the following proof does not have the unique signature property.

Proof: Starting from (Part 2 of) Theorem 6.5.1, we can use any 1-1 one-way function to obtain super-secure length-restricted one-time signature schemes. However, wishing to use arbitrary one-way functions, we will first show that universal one-way hashing functions can be used (instead of 1-1 one-way functions) to obtain super-secure length-restricted one-time signature schemes. Next, we will show that super-security is preserved by two transformations presented in Section 6.4: specifically, the transformation of *length-restricted* one-time signature schemes into one-time signature schemes (specifically, Construction 6.4.30), and the transformation of the latter to (full-fledged) signature schemes (i.e., Construction 6.4.16). Applying these transformations (to the first scheme), we obtained the desired super-secure signature scheme. Recall that Construction 6.4.30 also uses universal one-way hashing functions, but the latter can be constructed using any one-way function (cf. Theorem 6.4.29).[30]

Claim 6.5.2.1: If there exist universal one-way hashing functions then, for every polynomially-bounded $\ell : \mathbb{N} \to \mathbb{N}$, there exist super-secure ℓ-restricted one-time signature schemes.

Proof sketch: We modify Construction 6.4.4 by using universal one-way hashing functions (UOWHFs) instead of one-way functions. Specifically, for each preimage placed in the signing-key, we select at random and independently a UOWHF, and place its description both in the signing and verification keys. That is,

[30] We comment that a simpler proof suffices in case we are willing to use a one-way permutation (rather than an arbitrary one-way function). In this case, we can start from (Part 2 of) Theorem 6.5.1 (rather than prove Claim 6.5.2.1), and use Theorem 6.4.28 (rather than Theorem 6.4.29, which has a more complicated proof).

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 UOWHFs $h_1^0, h_1^1, ..., h_{\ell(n)}^0, h_{\ell(n)}^1$, and compute $v_i^j = h_i^j(s_i^j)$, for $i = 1, ..., \ell(n)$ and $j = 0, 1$. We let $\overline{s} = ((s_1^0, s_1^1), ..., (s_{\ell(n)}^0, s_{\ell(n)}^1))$, $\overline{h} = ((h_1^0, h_1^1), ..., (h_{\ell(n)}^0, h_{\ell(n)}^1))$, and $\overline{v} = ((v_1^0, v_1^1), ..., (v_{\ell(n)}^0, v_{\ell(n)}^1))$, and output the key-pair $(s, v) = ((\overline{h}, \overline{s}), (\overline{h}, \overline{v}))$ (or, actually, we may set $(s, v) = (\overline{s}, (\overline{h}, \overline{v}))$). Signing and verification are modified accordingly; that is, signing $\sigma_1 \cdots \sigma_\ell$ amounts to handing $(s_1^{\sigma_1}, ..., s_\ell^{\sigma_\ell})$, whereas $(\beta_1, ..., \beta_\ell)$ is accepted as a valid signature of $\sigma_1 \cdots \sigma_\ell$ (w.r.t the verification-key v) if and only of $h_i^{\sigma_i}(\beta_i) = v_i^{\sigma_i}$ for every i. In order to show that the resulting scheme is super-secure under a chosen *one*-message attack, we adapt the proof of Proposition 6.4.5. Specifically, fixing such an attacker A, we consider the event in which A violated the super-security of the scheme. There are two cases to consider:

1. The valid signature formed by A is to the same document for which A has obtained a different signature (via its single query). In this case, for at least one of the UOWHFs contained in the verification-key, we obtain a preimage that is different from the one contained in the signing-key. Adapting the construction presented in the proof of Proposition 6.4.5, we obtain (in this case) ability to form designated collisions (in contradiction to the UOWHF property). We stress that the preimages contained in the signing-key are selected independently of the description of the UOWHFs (because both are selected independently by the key-generation process). In fact, we obtain a designated collision for a uniformly selected preimage.

2. The valid signature formed by A is to a document that is different from the one for which A has obtained a signature (via its single query). In this case, the proof of Proposition 6.4.5 yields ability to invert a randomly selected UOWHF (on a randomly selected image), which contradicts the UOWHF property (as shown in Exercise 15).

Thus, in both cases we derive a contradiction, and the claim follows. □

Claim 6.5.2.2: Construction 6.4.30, when applied to a super-secure length-restricted signature scheme yields a super-secure signature scheme. In case the length-restricted scheme is only super-secure under a chosen one-message attack, the same holds for the the resulting (length-unrestricted) scheme.

Proof sketch: We follow the proof of Proposition 6.4.31, and use the same construction of a forger for the length-restricted scheme (based on the forger for the complex scheme). Furthermore, we consider the two forgery cases analyzed in the proof of Proposition 6.4.31:[31]

[31] Recall that (α, β) denotes the document-signature pair output by the original forger (i.e., for the complex scheme), whereas $(\alpha^{(i)}, \beta^{(i)})$ denotes the i^{th} query-answer pair (to that scheme). The document-signature pair that we output (as a candidate forgery w.r.t length-restricted scheme) is (α_2, β_2), where $\alpha_2 \stackrel{\text{def}}{=} (\beta_1, h_{\beta_1}(\alpha))$ and $\beta = (\beta_1, \beta_2)$. Recall that a generic valid document-signature for the complex scheme has the form (α', β'), where $\beta' = (\beta_1', \beta_2')$ satisfies $V_v((\beta_1', h_{\beta_1'}(\alpha')), \beta_2') = 1$.

Case 1: $(\beta_1, h_{\beta_1}(\alpha)) \neq (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$ for all i's. In this case, the analysis is exactly as in the original proof. Note that it does not matter whether or not $\alpha \neq \alpha^{(i)}$, since in both subcases we obtain a valid signature for a new string with respect to the length-restricted signature scheme. Thus, in this case, we derive a violation of the (ordinary) security of the length-restricted scheme.

Case 2: $(\beta_1, h_{\beta_1}(\alpha)) = (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$ for some i. The case $\alpha \neq \alpha^{(i)}$ was handled in the original proof (by showing that it yields a designated collision (under $h_{\beta_1^{(i)}}$ which is supposedly a UOWHF)), so here we only handle the case $\alpha = \alpha^{(i)}$. Now, suppose that super-security of the complex scheme was violated; that is, $(\beta_1, \beta_2) \neq (\beta_1^{(i)}, \beta_2^{(i)})$. Then, by the case hypothesis (which implies $\beta_1 = \beta_1^{(i)}$), it must be that $\beta_2 \neq \beta_2^{(i)}$. This means that we derive a violation of the *super*-security of length-restricted scheme, because β_2 is a different valid S_s-signature of $(\beta_1, h_{\beta_1}(\alpha)) = (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$.

> Actually, we have to consider all i's for which $(\beta_1, h_{\beta_1}(\alpha)) = (\beta_1^{(i)}, h_{\beta_1^{(i)}}(\alpha^{(i)}))$ holds, and observe that violation of super-security for the complex scheme means that β_2 must be different from each of the corresponding $\beta_2^{(i)}$'s. Alternatively, we may first prove that, with overwhelmingly high probability, all $\beta_1^{(i)}$'s must be distinct.

Thus, in both cases we reach a contradiction to the super-security of the length-restricted signature scheme, which establishes our claim that the general signature scheme must be super-secure. We stress that, like in Proposition 6.4.31, the above proof establishes that super-security for one-time attacks is preserved too (because the constructed forger makes a single query per each query made by the original forger). □

Claim 6.5.2.3: Construction 6.4.16, when applied to super-secure one-time signature schemes yields super-secure signature schemes.

Proof sketch: We follow the proof of Proposition 6.4.17, which actually means following the proof of Proposition 6.4.15. Specifically, we use the same construction of a forger for the one-time scheme (based on the forger for the complex scheme). Furthermore, we consider the two forgery cases analyzed in the proof of Proposition 6.4.15:[32]

1. The *first case* is when the forged signature for the complex (general signature) scheme (G', S', V') contains a signature relative to an instance of the one-time scheme (G, S, V) associated with a leaf that has been authenticated in an answer given to some signing-query. If no oracle answer has used the instance associated with this leaf then (as in the proof of

[32] Recall that forging a signature for the general scheme requires either using an authentication path supplied by the (general) signing-oracle or producing an authentication path different from all paths supplied by the (general) signing-signer.

Proposition 6.4.15) we obtain (ordinary) forgery with respect to the instance of (G, S, V) associated with the leaf (without making any query to that instance of the one-time scheme). Otherwise, by the case hypothesis, the forged document-signature pair differs from the query-answer pair that used the same leaf. The difference is either in the document part or in the part of the complex-signature that corresponds to the one-time signature produced at the leaf. In both subcases this yields violation of the super-security of the instance of (G, S, V) associated with that leaf. Specifically, in the first subcase we obtain a one-time signature to a different document (i.e., violation of ordinary security), whereas in the second subcase we obtain a different one-time signature to the same document (i.e., only a violation of super-security). We stress that, in both subcases, the violating signature is obtained after making a single query to the instance of (G, S, V) associated with that leaf.

2. 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, we obtain violation of the ordinary (one-time) security of the scheme (G, S, V), exactly as in the original proof of Proposition 6.4.15. We stress that in this case (regardless of which document is authenticated by the leaf), an internal node authenticates data that is different from the data authenticated by the signing-oracle, and thus we obtain forgery via a one-message attack on the instance of (G, S, V) associated with this internal node.

Thus, in both cases we reach a contradiction to the super-security of the one-time signature scheme, which establishes our claim that the general signature scheme must be super-secure. □

Combining the three claims (and recalling that universal one-way hashing functions can be constructed using any one-way function (cf. Theorem 6.4.29)), the theorem follows. ■

6.5.3 Off-line/on-line signing

Loosely speaking, we say that a signature scheme (G, S, V) (either a private-key or a public-key one) has an off-line/on-line signing process if signatures are produced in two steps, where the first step is independent of the actual message to be signed. That is, the computation of $S_s(\alpha)$ can be decoupled into two steps, performed by *randomized* algorithms that are denoted $S^{\rm off}$ and $S^{\rm on}$ respectively such that $S_s(\alpha) \leftarrow S_s^{\rm on}(\alpha, S^{\rm off}(s))$. Thus, one may prepare (or precompute) $S^{\rm off}(s)$ before the document is known (i.e., off-line), and produce the actual signature (on-line) once the document α is presented is produced (by invoking algorithm $S^{\rm on}$ on input $\alpha, S^{\rm off}(s)$). This yields improvement in on-line response-time to signature requests, provided that $S^{\rm on}$ is significantly faster that S itself. This improvement is worthwhile in many natural settings in which on-line response-time is more important than off-line processing time.

We stress that S^{off} must be randomized (as otherwise $S^{\text{off}}(s)$ can be incorporated in the signing-key). Indeed, one may view algorithm S^{off} as an extension of the key-generation algorithm that produces *random* extensions of the signing-key on the fly (i.e., after the verification-key was already determined). We stress that algorithm S^{off} is invoked once per each document to be signed, but this invocation can take place at any time and even before the document to be signed is even determined. (In contrast, it may be insecure to re-use the result obtained from S^{off} for two different signatures.)

Can secure signature schemes employ meaningful off-line/on-line signing algorithms? Of course, any algorithm can be vacuously decoupled into two steps, but we are only interested in meaningful decouplings in which the off-line step takes most of the computational load. Interestingly, schemes based on the refreshing paradigm (cf. Section 6.4.2.1) lend themselves to such a decoupling. Specifically, in Construction 6.4.16, only the last step in the signing process depends on the actual document (and needs to be performed on-line). Furthermore, this last step amounts to applying the signing algorithm of a one-time signature scheme, which is typically much faster than all the other steps (which can be performed off-line).[33]

6.5.4 Incremental signatures

Loosely speaking, we say that a signature scheme (G, S, V) (either a private-key or a public-key one) has an incremental signing process if the signing process can be sped-up when given a valid signature to a (textually) related document. The actual definition refers to a set of text editing operations such as *delete word* and *insert word* (where more powerful operations like *cutting* a document into two parts and *pasting* two documents may be supported two). Specifically, one may require that given a document-signature pair, (α, β), a sequence of edit operations (i.e., specifying the operation type and its location), and the signing-key one may modify β into a valid signature for the modified document in time proportional to the number of edit operations (and not to $|\alpha|$). Indeed, here time is measured in a direct-access model of computation. Of course, the time saving on the signing side should not come at the expense of a significant increase in verification time. In particular, verification time should only depend on the length of the final document (and not on the number of edit operations).

An incremental signing process is beneficial in settings where one needs to sign many textually related documents (e.g., in simple contracts much of the text

[33] When plugging-in the one-time signature scheme suggested in Proposition 6.4.7, producing one-time signatures amounts to applying a collision-free hashing function and outputting corresponding parts of the signing-key. This is all that needs to be performed in the on-line step. In contrast, the off-line steps calls for n applications of a pseudorandom function, n applications of the key-generation algorithm of the one-time signature scheme, and n applications of the signing algorithm of the one-time signature scheme.

is almost identical and edit changes refer to the party's specific details as well as to specific clauses that are modified from their standard form in order to meet the party's specific needs). In some cases the *privacy of the edit sequence* may be of concern; that is, one may require that the final signature be distributed in a way that only depends on the final document (rather than depend also on documents that "contributed" signatures to the process of generating the final signature).

Can secure signature schemes employ a meaningful incremental signing process? Here meaningful refers to the set of supported text-modification operations. The answer is affirmative, and furthermore these schemes may even protect the privacy of the edit sequence. Below, we refer to edit operations that delete/insert fix-length bit-strings called blocks from/to a document (as well as to the cut and paste operations mentioned above).

Theorem 6.5.3 (secure schemes with incremental signing process):

1. *Assuming the existence of one-way functions, there exist secure message authentication schemes having an incremental signing process that supports block deletion and insertion. Furthermore, the scheme uses a fixed-length authentication tag.*
2. *Assuming the existence of one-way functions, there exist secure* (private-key and public-key) *signature schemes having an incremental signing process that supports block deletion and insertion as well as cut and paste.*

Furthermore, in both parts, the resulting schemes protect the privacy of the edit sequence.

Part 1 is proved by using a variant on an efficient message authentication scheme that is related to the schemes presented in Section 6.3.1. Part 2 is proved by using an arbitrary secure (private-key or public-key) signature scheme that produces n-bit long signatures to $O(n)$-bit long strings, where n is the security parameter. (Indeed, the scheme need only be secure in the $O(n)$-restricted sense.) The document is stored in the leaves of a 2–3 tree,[34] and the signature essentially consists of the tags of all internal nodes, where each internal node is tagged by applying the basic signature scheme to the tags of its children. One important observation is that a 2–3 tree supports the said operations while incurring only

[34] A 2–3 tree is a balanced tree in which each internal node has either 2 or 3 children. Such trees support insert and delete (of a single symbol/leaf) in logarithmically many operations. To insert a leaf (in a depth d tree), add it as a child to the suitable level $d-1$ vertex, denoted v. In case the resulting children-degree of v is 4, split v (evenly) into two vertices such that both the resulting vertices are children of v's parent. The parent may be split so too, and so on until one gets to the root. If the root needs to be split then the height of the tree is incremented. To delete a leaf, we apply an analogous procedure. Namely, if the resulting parent and its siblings have total children-degree at least 4 then we rearrange these children so that each of the resulting parent nodes has children-degree either 2 or 3. In case the total children-degree is at most 3, we merge the parent and its sibling to one vertex and turn to its parent. Cutting and pasting of (sub)trees can be performed analogously.

a logarithmic (in its size) cost; that is, modifying only the links of logarithmic many nodes in the tree. Thus, only the tags of these nodes and their ancestors in the tree needs to be modified in order to form the correspondingly modified signature. (Privacy of the edit sequence is obtained by randomizing the standard modification procedure for 2–3 trees.) By analogy to Construction 6.2.13 (and Proposition 6.2.14), the incremental signature scheme is secure.

6.5.5 Fail-stop signatures

Loosely speaking, a fail-stop signature scheme is a signature scheme augmented by a (non-interactive) proof system that allows the legitimate signer to prove to anybody that a particular (document,signature)-pair was not generated by him/her. Actually, key-generation involves interaction with an *administrating entity* (which publicizes the resulting verification-keys), rather than just having the user publicize his/her verification-key. In addition, we allow memory-dependent signing procedures (as in Definition 6.4.13).[35] The system guarantees the following four properties, where the first two properties are the standard ones:

1. *Proper operation*: In case the user is honest, the signatures produced by it will pass the verification procedure (with respect to the corresponding verification-key).

2. *Infeasibility of forgery*: In case the user is honest, forgery is infeasible in the standard sense. That is, every feasible chosen message attack may succeed (to generate a valid signature to a new message) only with negligible probability.

3. *Revocation of forged signatures*: In case the user is honest, it can prove that forgery has been committed (in case it was indeed committed). That is, for every chosen message attack (even a computationally-unbounded one)[36] that produces a valid signature to a new message, except for with negligible probability, the user can convince anyone (which knows the verification-key) that this valid signature was forged (i.e., produced by somebody else). The probability is taken over the actions of the (computationally-unbounded) adversary committing forgery.

4. *Infeasibility of revoking unforged signatures*: It is infeasible for a user to create a valid signature and later convince anybody that this signature was forged (i.e., produced by somebody else). Indeed, it is possible (but not feasible) for a user to cheat here.

Furthermore, Property 3 (i.e., revocation of forged signatures) holds also in case the administrating entity participates in the forgery and even if it behaves

[35] Allowing memory-dependent signing is essential to the existence of secure fail-stop signature schemes; see Exercise 21.

[36] It seems reasonable to restrict such adversaries to polynomially-many signing requests.

improperly at the key-generation stage. (In contrast, the other items hold *only* if the administrating entity behaves properly during the key-generation stage.)

To summarize, fail-stop signature schemes allow to prove that forgery has occurred, and so offer an information-theoretic security guarantee to the potential signers (yet the guarantee to potential signature-recipients is only a computational one).[37] In contrast, when following the standard semantics of signature schemes, the potential signers have only a computational security guarantee and the signature recipients have an absolute guarantee: whenever the verification algorithm accepts a signature, it is by definition an unrevocable one.

Do secure fail-stop signature schemes exist? Assuming the intractability of either the Discrete Logarithm Problem or of integer factorization, the answer is affirmative. Indeed, in fail-stop signature schemes, each document must have super-polynomially many possible valid signatures (with respect to the publically known verification-key), but only a negligible fraction of these will be (properly) produced by the legitimate signer (who knows a corresponding signing-key, which is not uniquely determined by the verification-key). Furthermore, any strategy (even an infeasible one), is unlikely to generate signatures corresponding to the signing-key. On the other hand, it is infeasible given one signing-key to produce valid signatures (i.e., w.r.t the verification-key) that do not correspond to the proper signing with this signing-key.

6.6 Miscellaneous

6.6.1 On Using Signature Schemes

Once defined and constructed, signature schemes may be (and are actually) used as building blocks towards various goals that are different from the original motivation. Still, the original motivation (i.e., reliable communication of information) is of great importance, and in this subsection we discuss several issues regarding the use of signature schemes towards achieving it. The discussion is analogous to a similar discussion conducted in Section 5.5.1, but the analogous issues discussed here are even more severe.

Using private-key schemes – the key exchange problem. As discussed in Section 6.1, using a private-key signature scheme (i.e., a message authentication scheme) requires the communicating parties to share a secret key. This key can be generated by one party and secretly communicated to the other party by an alternative (expensive) secure and reliable channel. Often, a preferable solution consists of employing a *key-exchange* (or rather *key-generation*) protocol, which is executed over the standard (unreliable) communication channel.

[37] The above refers to the natural convention by which a proof of forgery frees the signer of any obligations implied by the document. In this case, when accepting a valid signature the recipient is only guaranteed that it is infeasible for the signer to revoke the signature.

We stress that here (unlike in Section 5.5.1) we must consider active adversaries. Consequently, the focus should be on key-exchange protocols that are secure against active adversaries and are called unauthenticated key-exchange protocols (because the messages received over the channel are not necessarily authentic). Such protocols are too complex to be treated in this section, and the interested reader is referred to [30, 31, 18].

Using state-dependent message authentication schemes. In many communication settings it is reasonable to assume that the authentication device may maintain (and modify) a state (e.g., a counter or a clock). Furthermore, in many applications, a changing state (e.g., a clock) must be employed anyhow in order to prevent reply of old messages (i.e., each message will be authenticated along with its transmission time). In such cases, state-dependent schemes as discussed in Section 6.3.2 may be preferable. (See further discussion in Section 6.3.2 and analogous discussion in Section 5.5.1.)

Using signature schemes – public-key infrastructure. The standard use of (public-key) signature schemes in real-life applications requires a mechanism for providing the verifiers with the signer's authentic verification-key. In small systems, one may assume that each user holds a local record of the verification-keys of all other users. However, this is not realistic in large-scale systems, and so the verifier must obtain the relevant verification-key on-the-fly in a "reliable" way (i.e., typically, certified by some trusted authority). In most theoretical work, one assumes that the verification-keys are posted and can be retrieved from a public-file that is maintained by a trusted party (which makes sure that each user can post only verification-keys bearing its own identity). In abstract terms, such trusted party may provide each user with a (signed) certificate stating the authenticity of the user's verification-key. In practice, maintaining such a public-file (and handling such certificates) is a major problem, and mechanisms that implement this abstraction are typically referred to by the generic term "public-key infrastructure (PKI)". For a discussion of the practical problems regarding PKI deployment see, e.g., [180, Chap. 13].

6.6.2 On Information Theoretic Security

In contrast to the bulk of our treatment, which focuses on computationally-bounded adversaries, in this section we consider computationally-unbounded adversaries. Specifically, we consider computationally-unbounded chosen message attacks, but do bound (as usual, by an unknown polynomial) the total number of bits in the signing-queries made by such attackers. We call a (private-key or public-key) signature scheme perfectly-secure (or information-theoretically secure) if even such computationally-unbounded attackers may succeed (in forgery) only with negligible probability.

It is easy to see that no (public-key) signature scheme may be perfectly-secure, *not even in a length-restricted one-time sense.* The reason is that a computationally-unbounded adversary that is given a verification-key can find (without making any queries) a corresponding signing-key, which allows it to forge signatures to any message of its choice.

In contrast, *restricted types* of message authentication schemes (i.e., private-key signature schemes) may be perfectly-secure. Specifically, given any polynomial bound on the total number of messages to be authenticated, one may construct a corresponding state-based perfectly-secure message authentication scheme. In fact, a variant of Construction 6.3.10 will do, where a truly random one-time pad is used instead of the pseudorandom sequence generated using the next-step function g. Indeed, this one-time pad will be part of the key, which in turn must be longer than the total number of messages to be authenticated. We comment that the use of a state is essential for allowing several messages to be authenticated (in a perfectly-secure manner). (Proofs of both statements can be derived following the ideas underlying Exercise 8.2.)

6.6.3 On Popular Schemes

The reader may note that we have avoided the presentation of several popular signature schemes (i.e., public-key ones). As noted in Section 6.1.4.3, some of these schemes (e.g., RSA [216] and DSS [192]) seem to satisfy some weak (i.e., weaker than Definition 6.1.2) notions of security. Variants of these schemes are proven to be secure *in the random oracle model*, provided some standard intractability assumptions hold (cf, e.g., [32]). However, we are not satisfied with either of these types of results, and articulate our opinion next.

On using weaker definitions. We distinguish between weak definitions that make clear reference to the abilities of the adversary (e.g., one-message attacks, length-restricted message attacks) and weak notions that make hidden and unspecified assumptions regarding what may be beneficial to the adversary (e.g., "forgery of signatures for meaningful documents"). In our opinion, the fact that the hidden assumptions often "feel right" makes them even more dangerous, because it means that they are never seriously considered (and not even formulated). For example, it is often said that *existential forgery* (see Section 6.1.3) is "merely of theoretical concern", but these claims are never supported by any evidence or by a specification of the types of forgery that are of "real practical concern". Furthermore, a few years later, one learns that this "merely theoretical" issue yields a real security breach in some important applications. Still, weak definition of security may make sense, provided that they are clearly stated and that one realizes their limitations (i.e., "non-generality"). Since this book focuses on generally-applicable definitions, we chose not to discuss such weaker notions of security and not to present schemes that can be evaluated only with respect to these weak notion.

On the Random Oracle Methodology. The *Random Oracle Methodology* [95, 29] consists of two steps: First, one designs an *ideal* system in which all parties (including the adversary) have oracle access to a truly random function, and proves this *ideal* system to be secure (i.e., one typically says that the system is *secure in the random oracle model*). Next, one replaces the random oracle by a "good cryptographic hashing function", providing all parties (including the adversary) with the succinct description of this function, and hopes that the resulting (actual) scheme is secure.[38] We warn that this hope has no justification. Furthermore, there exist encryption and signature schemes that are secure in the Random Oracle Model, but replacing the random function (used in them) by *any* function ensemble yields a totally insecure scheme (cf., [59]).

6.6.4 Historical Notes

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 current section is organized accordingly.

6.6.4.1 Signature Schemes

The notion of a (public-key) signature scheme was introduced by Diffie and Hellman [78], who also suggested to implement it using trapdoor permutations. Concrete implementations were suggested by Rivest, Shamir and Adleman [216] and by Rabin [211]. 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 [145], but their definition is weaker than the one followed in our text. (Specifically, the adversary's queries in the definition of [145] 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 [145] is significantly stronger than all security notions considered before [145].

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 [143]. 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 [145] were inherently memory-dependent, the scheme of [143] has a "memoryless" variant (cf. [105] and [143]).

[38] Recall that, in contrast, the methodology of Section 3.6.3 (which is applied often in the current chapter) refers to a situation in which the adversary does not have direct oracle access to the random function, and does not obtain the description of the pseudorandom function used in the latter implementation.

Following Goldwasser, Micali and Rivest [143], research has focused on constructing secure signature schemes under weaker assumptions. In fact, as noted in [143], their construction of secure signature schemes can be carried out using any collection of *claw-free, trapdoor* permutation pairs. The claw-free requirement was omitted in [28], whereas the seemingly more fundamental trapdoor requirement was omitted by Naor and Yung [198]. Finally, Rompel showed that one may use arbitrary one-way functions rather one-way permutations [217], and thus established Theorem 6.4.1. The progress briefly summarized above was enabled by the use of many important ideas and paradigms, some of them were introduced in that body of work and some were "only" revisited and properly formalized. Specifically, we refer to the introduction of the refreshing paradigm in [143], the use of authentication trees (cf., [182, 183] and [143]), the use of the hash-and-sign paradigm (rigorously analyzed in [72]), the introduction of Universal One-Way Hash Functions (and the adaptation of the hash-and-sign paradigm to them) in [198], and the use of one-time signature schemes (cf., [210]).

We comment that our presentation of the construction of signature schemes is different from the one given in any of the above cited papers. Specifically, the main part of Section 6.4 (i.e., Sections 6.4.1 and 6.4.2) is based on a variant of the signature scheme of [198], in which collision-free hashing (cf. [72]) are used instead of universal one-way hashing (cf. [198]).

6.6.4.2 Message Authentication Schemes

Message authentication schemes were first discussed in the information theoretic setting, where a one-time pad was used. Such schemes were first suggested in [104], and further developed in [236]. The one-time pad can be implemented by a pseudorandom function (or a on-line pseudorandom generator), yielding only computational security, as we have done in Section 6.3.2. Specifically, Construction 6.3.10 is based on [163, 164]. In Section 6.3.1 we have followed a different paradigm that amounts to applying a pseudorandom function to the message (or its hashed-value), rather than using a pseudorandom function (or a on-line pseudorandom generator) to implement a one-time pad. This alternative paradigm is due to [119], and is followed in works such as [27, 24, 16]. Indeed, following this paradigm (and similarly to [27, 24, 16]), we have actually focused (in Section 6.3.1) on constructing generalized pseudorandom function ensembles (as in Definition 3.6.12), based on ordinary pseudorandom functions (as in Definition 3.6.4).

Collision-free hashing

Collision-free hashing was first defined in [72]. Construction 6.2.8 is also due to [72], with underlying principles that can be traced to [143]. Construction 6.2.11 is due to [73]. Construction 6.2.13 is due to [184].

On the additional properties

Unique signatures and super-security have been used in several works, but never extensively treated before. The notion of offline/online signature scheme was introduced (and first instantiated) in [86]. The notion of incremental cryptographic schemes (and in particular incremental signature schemes) was introduced and instantiated in [21, 22]. In particular, the incremental MAC of [22] (i.e., Part 1 of Theorem 6.5.3) builds on the message authentication scheme of [24], and the incremental signature scheme that protects the privacy of the edit sequence is due to [188] (building upon [22]). Fail-stop signatures were defined and constructed in [206].

6.6.5 Suggestion for Further Reading

As mentioned above, the work of Goldwasser, Micali and Rivest contains a comprehensive treatment of security notions for signature schemes [143]. 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 forgery 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 [143]. (Still, in some applications, weaker notions of security may suffice.) We stress that one may still benefit from the definitional part of [143], but the constructive part of [143] should be ignored since it is superseded by later work (on which our presentation is based).

Pfitzmann's book [207] 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 [207, Chap. 7–11], but much attention is given to the presentation of a general framework [207, Chap. 5] and to review of other "non-ordinary" schemes [207, Sec. 2.7 & 6.1].

As hinted in Section 6.6.4.2, our treatment of the construction of message authentication schemes is merely the tip of an iceberg. The interested reader is referred to [230, 163, 164, 40] for details on the "one-time pad" approach, and to [27, 24, 16, 17, 23, 7] for alternative approaches. Constructions and discussion of AXU hashing functions can be found in [163, 164].

The constructions of universal one-way hash functions presented in Section 6.4.3 use any one-way permutation, and do so in a generic way. The number of applications of the one-way permutation in these constructions is linearly related to the difference between the number of input and output bits in the hash function. In [103], it is shown that as far as generic (black-box) constructions go, this is essentially the best performance that one can hope for.

In continuation to the discussion in Section 6.4.2.4, we refer to reader to [82, 69], in which specific implementations (of a generalization) of Constructions 6.4.14

and 6.4.16 are presented. Specifically, these works utilize an authentication tree of large degree (rather than binary trees as in Section 6.4.2.2).

6.6.6 Open Problems

The known construction of signature schemes from *arbitrary* one-way functions [217] is merely a feasibility result. 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 important new paradigms.

6.6.7 Exercises

Exercise 1: *Deterministic Signing and Verification algorithms*:

1. Using a pseudorandom function ensembles, show how to transform any (private-key or public-key) signature scheme into one employing a deterministic signing algorithm.
2. Using a pseudorandom function ensembles, show how to transform any message authentication scheme into one employing deterministic signing and verifying algorithms.
3. Verify that all signature schemes presented in the current chapter employ a deterministic verification algorithm.

Guideline (for Part 1): Augment the signing-key with a description of a pseudorandom function, and apply this function to the string to be signed in order to extract the randomness used by the original signing algorithm.

Guideline (for Part 2): Analogous to Part 1. (Highlight your use of the private-key hypothesis.) Alternatively, see Exercise 2.

Exercise 2: *Canonical verification in the private-key version*: Show that, without loss of generality, the verification algorithm of a private-key signature scheme may consist of comparing the alleged signature to one produced by the verification algorithm itself (which does so exactly as the signing algorithm).

Why does this claim fail with respect to public-key schemes?

Guideline: Use Part 1 of Exercise 1, and conclude that the on a fixed input the signing algorithm always produces the same output. Use the fact that (by Exercise 8.2) the existence of message authentication schemes implies the existence of pseudorandom functions.

Exercise 3: *Augmented attacks in the private-key case*: In continuation to the discussion in Section 6.1.4.1, consider the definition of an augmented attack (on a private-key signature scheme) in which the adversary is allowed verification-queries.

1. Show that in case the signature scheme has (a deterministic verification algorithm and) unique valid signatures, it is secure against augmented attacks if and only if it is secure against ordinary attacks (as in Definition 6.1.2).
2. Assuming the existence of secure private-key signature schemes (as in Definition 6.1.2), present such a secure scheme that is insecure under augmented attacks.

Guideline (Part 1): Analyze the emulation outlined in Section 6.1.4.1. Specifically, ignoring the redundant verification-queries (for which the answer is determined by previous answers), consider the probability that the emulation has gambled correctly on all the verification-queries up-to (and including) the *first* such query that should be answered affirmatively.

Guideline (Part 2): Given any secure MAC (G, S, V), assume without loss of generality that in the key-pairs output by G the verification-key equals the signing-key. Consider the scheme (G', S', V) (with $G' = G$), where $S'_s(\alpha) = (S_s(\alpha), 0)$, $V'_v(\alpha, (\beta, 0)) = V_v(\alpha, \beta)$ and $V'_v(\alpha, (\beta, i, \sigma)) = 1$ if both $V_v(\alpha, \beta) = 1$ and the i^{th} bit of $s = v$ is σ. Prove that (G', S', V) is secure under ordinary attacks, and present an augmented attack that totally breaks it (i.e., obtains the signing-key).

Exercise 4: *The signature may reveal the document*: Both for private-key and public-key signature schemes, show that if such secure schemes exist then there exist secure signature schemes in which any valid signature to a message allows to efficiently recover the entire message.

Exercise 5: *On the triviality of some length-restricted signature schemes*:

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

Guideline (Part 2): 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 to any string of its choice.

Exercise 6: *Failure of Construction 6.2.3 in case $\ell(n) = O(\log n)$*: Show that if Construction 6.2.3 is used with logarithmically bounded ℓ then the resulting scheme is insecure.

> **Guideline:** Note that by asking for polynomially-many signatures, the adversary may obtain two S'_s-signatures that use the same (random) identifier. Specifically, consider making the queries $\alpha\alpha$, for all possible $\alpha \in \{0,1\}^{\ell(n)}$, and note that if $\alpha\alpha$ and $\alpha'\alpha'$ are S'_s-signed using the same identifier then we can derive a valid S'_s-signature to $\alpha\alpha'$.

Exercise 7: 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 8: Prove that the existence of secure message authentication schemes implies the existence of one-way functions. Specifically, let (G, S, V) be as in the hypothesis.

1. To simplify the following two items, show that, without loss of generality, $G(1^n)$ uses n coins and outputs a signing-key of length n and that $|S_s(\alpha)|$ is determined by $|s| + |\alpha|$.
2. Assume first that S is a *deterministic* signing algorithm. Prove that $f(r, \alpha_1, ..., \alpha_m) \stackrel{\text{def}}{=} (S_s(\alpha_1), ..., S_s(\alpha_m), \alpha_1, ..., \alpha_m)$ is a one-way function, where $s = G_1(r)$ is the signing-key generated with coins r, all α_i's are of length $n = |r|$ and $m = \Theta(n)$.
   Extend the proof to handle randomized signing algorithms.
3. Using the relation between pseudorandom functions (as in Definition 3.6.12) and one-way functions, the following provides an alternative proof for the special case of deterministic signing.[39]
   (Based on [197]): Consider the Boolean function ensemble $\{f_{s,r}\}_{s,r}$, where s is selected according to $G_1(1^n)$ and r is uniformly distributed over strings of length $|S_s(1^n)|$, defined such that $f_{s,r}(\alpha)$ equals the inner-product mod 2 of r and $S_s(\alpha)$. Prove that this ensemble is pseudorandom (as defined in Definition 3.6.12 for the case $r(n) = 1$).

> **Guideline (Part 2):** Note that the m signatures determine an r', which in turn determines a signing-key $s' = G_1(r')$ such that $S_s(\alpha) = S_{s'}(\alpha)$ for most $\alpha \in \{0,1\}^n$. (Note that s' does not necessarily equal s.) Show that this implies that ability to invert f yields ability to forger (under a chosen message attack). (Hint: use m random signing-queries to produce a

[39] Note that the functions in the ensemble have a sufficiently large domain. Thus, this pseudorandom function ensemble gives rise to a pseudorandom generator (analogously to Exercise 28 of Chapter 3), which in turn implies the existence of one-way functions.

random image of f.) The extension to randomized signing is obtained by augmenting the argument of the one-way function with the coins used by the m invocations of the signing algorithm.

Guideline (Part 3): Consider hybrid experiments such that in the ith hybrid the first i queries are answered by a truly random Boolean function and the rest are answered by a uniformly distributed $f_{s,r}$. (Note that it seems important to use this non-standard order of random versus pseudorandom answers.) Show that distinguishability of the ith and $i+1$st hybrids implies that a probabilistic polynomial-time machine can have a non-negligible advantage in the following game in which the machine is asked to select α, next $f_{s,r}$ is uniformly selected and the machine is given r as well as oracle access to S_s (but is not allowed the query α) and is asked to guess $f_{s,r}(\alpha)$. (Note that the particular order used allows to produce the rest of the hybrid when given this oracle access. On the other hand, it is important to hand r only after the machine has selected α; see [197].) At this point, one may apply the proof of Theorem 2.5.2, and deduce that the said machine can construct $S_s(\alpha)$ with non-negligible probability, in contradiction to the security of the MAC.

Exercise 9: 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 producing that output.

Exercise 10: *On perfectly-secure one-time message authentication (MAC) schemes*: By perfect (or information-theoretic) security we mean that even computationally-unbounded chosen message attacks may succeed (in forgery) only with negligible probability.

Define perfect (or information-theoretic) security for one-time MACs and length-restricted one-time MACs. (Be sure to bound the length of documents (e.g., by some super-polynomial function) also in the unrestricted case.)

Prove the following, without relying on any (intractability) assumptions (which are anyhow useless in the information-theoretic context):

1. For any polynomially-bounded and polynomial-time computable function $\ell : \mathbb{N} \to \mathbb{N}$, perfectly-secure ℓ-restricted one-time MACs can be trivially constructed.
2. Using a suitable AXU family of hashing functions, present a construction of a perfectly-secure one-time MAC. Furthermore, present such a MAC in which the authentication-tags have fixed length (i.e., depending on the length of the key but not on the length of the message being authenticated).

3. Show that any perfectly-secure one-time MAC that utilizes fixed length authentication-tags and a deterministic signing algorithm yields a generalized hashing ensembles with negligible collision probability. Specifically, for any polynomial p, this ensembles has a $(p, 1/p)$-collision property.

Guideline: For Part 1, combine the ideas underlying Exercise 5 and Construction 6.4.4. For Part 2, use the ideas underlying Construction 6.3.10 and the proof of Proposition 6.3.11. For Part 3, given a MAC (G, S, V) as in the claim, consider the functions $h_s(x) \stackrel{\text{def}}{=} S_s(x)$, where $s \leftarrow G_1(1^n)$.

Exercise 11: In contrast to Exercise 10, prove that the existence of secure *one-time signature schemes* implies the existence of one-way functions. Furthermore, prove that this holds even for 1-restricted signature schemes that are secure (only) under attacks that make no signing-queries.

Guideline: See guideline for Item 2 in Exercise 5.

Exercise 12: 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 13: In contrast to Exercise 4, show that if secure message authentication schemes exist then there exist such schemes in which it is infeasible (for a party not knowing the key) to extract from the signature any partial information about the message (except for the message length). (Indeed, privacy of the message is formulated as the definition of semantic security of encryption schemes; see Chapter 5.)

Guideline: Combine a message authentication scheme with an adequate private-key encryption scheme. Refer to issues such as the type of security required of the encryption scheme, and why the hypothesis yields the existence of the ingredients used in the construction.

Exercise 14: In continuation to Exercise 13, show that if there exist collision-free hashing functions then there exist message authentication schemes in which it is infeasible (for a party not knowing the key) to extract from the signature any partial information about the message (including the message length). How come we can hide the message length in this context, whereas we cannot do this in the context of encryption schemes?

Guideline: Combine a message authentication scheme having fixed length signatures with an adequate private-key encryption scheme. Again, refer to issues as in Exercise 13.

Exercise 15: Prove that the existence of collections of UOWHF implies the existence of one-way functions. Furthermore, show that uniformly chosen functions in any collection of UOWHFs are hard to invert (in the sense of Definition 2.4.3).

Guideline: Note that the guidelines provided in Exercise 12 can be modified to fit the current context. Specifically, the collision-forming algorithm is given uniformly distributed r and x, and invokes the inverter on input $(r, h_r(x))$. Note that the furthermore clause is implicit in the proof.

Exercise 16: 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 $f'_s(x) = (0, f_s(x))$ if the $|s|$-bit long prefix of x is different from s, and $f'_s(sx') = (1, s)$ otherwise. Clearly, F' is not collision-free. Show that F' remains universal one-way hashing.

Exercise 17: Show that for every finite family of functions H, there exists $x \neq y$ such that $h(x) = h(y)$ for every $h \in H$. Furthermore, for $H = \{h : \{0,1\}^* \to \{0,1\}^m\}$, show that this holds for $|x|, |y| \leq m \cdot |H|$.

Guideline: Consider the mapping $x \mapsto (h_1(x), ..., h_t(x))$, where $H = \{h_i\}_{i=1}^t$. Since the number of possible images is at most $(2^m)^t$, we get a collision as soon as we consider more than 2^{mt} preimages.

Exercise 18: *Constructions of Hashing Families with Bounded Collision Probability*: In continuation to Exercise 22.2 in Chapter 3, consider the set of functions S_ℓ^m associated with ℓ-by-m Toeplitz matrix; that is $h_T(x) = Tx$, where $T = (T_{i,j})$ is a Toeplitz matrix (i.e., $T_{i,j} = T_{i+1,j+1}$ for all i, j). Show that this family has collision probability 2^{-m}. (Note that each ℓ-by-m Toeplitz matrix is specified using $\ell + m - 1$ bits.)

Guideline: Note that we have eliminated the shifting vector b used in Exercise 22.2 of Chapter 3, but this does not effect the relevant analysis.

Exercise 19: *Constructions of Generalized Hashing Families with Bounded Collision Property*: (See definition in Section 6.3.1.3.)

1. Using a tree-hashing scheme as in Construction 6.2.13, construct a generalized hashing ensemble with a $(f, 1/f)$-collision property, where $f(n) = 2^{\varepsilon n^\varepsilon}$ for some $\varepsilon > 0$.

2. (By Hugo Krawczyk): Show that the block-chaining method (as in Construction 6.2.11) fails in the current context. That is, there exists a hashing ensemble $\{h_r : \{0,1\}^{2m(|r|)} \to \{0,1\}^{m(|r|)}\}$ with negligible collision probability such that applying Construction 6.2.11 to it (even with three blocks) yields an ensemble with high collision probability.

Guideline (Part 1): Let $\{h_r : \{0,1\}^{2m(|r|)} \to \{0,1\}^{m(|r|)}\}$, be a hashing ensemble with collision probability $\mathtt{cp}$. Recall that such ensembles with $m(n) = n/3$ and $\mathtt{cp}(n) = 2^{-m(n)}$ can be constructed (see Exercise 18). Then, consider the function ensemble $\{h_{r_1,...,r_{m(n)}} : \{0,1\}^* \to \{0,1\}^{2m(n)}\}_{n\in\mathbb{N}}$, where all r_i's are of length n, such that $h_{r_1,...,r_{m(n)}}(x)$ is defined as follows

1. As in Construction 6.2.13, break x into $t \stackrel{\text{def}}{=} 2^{\lceil\log_2(|x|/m(n))\rceil}$ consecutive blocks, denoted $x_1, ..., x_t$, and let $d = \log_2 t$.
2. Let $i = 1, ..., t$, let $y_{d,i} \stackrel{\text{def}}{=} x_i$. For $j = d-1, ..., 1, 0$ and $i = 1, ..., 2^j$, let $y_{j,i} = h_{r_j}(y_{j+1,2i-1}y_{j+1,2i})$. The hash value equals $(y_{0,1}, |x|)$.

The above functions have description length $N \stackrel{\text{def}}{=} m(n) \cdot n$ and map strings of length smaller than $2^{m(n)}$ to strings of length $2m(n)$. It is easy to bound the collision probability (for strings of equal length) by the probability of collision occuring in each of the levels of the tree. In fact, for $x_1 \cdots x_t \neq x'_1 \cdots x'_t$ such that $x_i \neq x'_i$, it suffices to bound the sum of the probabilities that $y_{j,\lceil i/2^{d-j}\rceil} = y'_{j,\lceil i/2^{d-j}\rceil}$ holds (given that $y_{j+1,\lceil i/2^{d-(j+1)}\rceil} \neq y'_{j+1,\lceil i/2^{d-(j+1)}\rceil}$) for $j = d-1, ..., 1, 0$. Thus, this generalized hashing ensemble has a (ℓ, ϵ)-collision property, where $\ell(N) = 2^{m(n)} - 1$ and $\epsilon(N) = m(n) \cdot \mathtt{cp}(n)$. Recalling that we may use $m(n) = n/3$ and $\mathtt{cp}(n) = 2^{-m(n)}$, we obtain (using $N = n^2/3$), $\ell(N) = 2^{(N/3)^{1/2}} - 1 > 2^{(N/4)^{1/2}}$ and $\epsilon(N) < (N/\ell(N)) < 2^{-(N/4)^{1/2}}$ (as desired).

Guideline (Part 2): Given a hashing family as in the hypothesis, modify it into $\{h'_{r,s} : \{0,1\}^{2m} \to \{0,1\}^m\}$, such that $h'_{r,s}(0^{2m}) = s$, $h'_{r,s}(s\sigma^m) = 0^m$ for both $\sigma \in \{0,1\}$, and $h'_{r,s}(x) = h_r(x)$ for all other x's. Note that the new family maintains the collision probability of the original one up-to an additive term of $O(2^{-m})$. On the other hand, for both $\sigma \in \{0,1\}$, it holds that $h'_{r,s}(h'_{r,s}(0^{2m})\sigma^m) = h'_{r,s}(s\sigma^m) = 0^m$.

Exercise 20: *Additional properties required in Proposition 6.4.21*: In continuation to Exercise 23 of Chapter 3, show that the said function ensemble satisfies the following two properties:

1. All but a negligible fraction of the functions in S_n^{n-1} are 2-to-1.
2. 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 $\{s \in S_n^{n-1} : h_s(y_i) = z_i \; \forall i \in \{1,2\}\}$.

Guideline: Recall that 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 $\mathrm{GF}(2^n)$. The first condition follows by observing that the function $h_{a,b}$ is 2-to-1 if and only if

$a \neq 0$. The second condition follows by observing that $h_{a,b}(y_i) = z_i$ if and only if $ay_i + b = v_i$ for some v_i that is a single-bit extension of z_i. Thus, generating a pair (a,b) such that $h_{a,b}(y_i) = z_i$ for both i's, amounts to selecting random single-bit extensions v_i's, and (assuming $y_1 \neq y_2$) solving the system $\{ay_i + b = v_i\}_{i=1,2}$ (for the variables a and b).

Exercise 21: *Fail-stop signatures require a memory-dependent signing process*: In continuation to Section 6.5.5, prove that a secure fail-stop signature scheme must employ a memory-dependent signing process (as in Definition 6.4.13).

Guideline: Suppose towards the contradiction that there exist a secure memoryless fail-stop signature scheme. For every signing-key $s \in \{0,1\}^n$, consider the randomized process P_s in which one first selects uniformly $x \in \{0,1\}^n$, produces a (random) signature $y \leftarrow S_s(x)$, and outputs the pair (x, y). Show that, given polynomially-many samples of P_s, one can find (in exponential time) a string $s' \in \{0,1\}^n$ such that with probability at least 0.99 the statistical distance between P_s and $P_{s'}$ is at most 0.01. Thus, a computationally unbounded adversary making polynomially-many signing queries, can find a signing-key that typically produces the same signatures as the true signer. It follows that either these signatures cannot be revoked or that the user may also revoke its own signatures.

Author's Note: *First draft written mainly in May 2000. Major revision completed in Feb. 2002.*