SIAM J. COMPUT.
Vol. 26, No. 2, pp. 291–330, April 1997

FAIL-STOP SIGNATURES*

TORBEN PRYDS PEDERSEN† AND BIRGIT PFITZMANN‡

Abstract. Fail-stop signatures can briefly be characterized as digital signatures that allow the signer to prove that a given forged signature is indeed a forgery. After such a proof has been published, the system can be stopped. This type of security is strictly stronger than that achievable with ordinary digital signatures as introduced by Diffie and Hellman in 1976 and formally defined by Goldwasser, Micali, and Rivest in 1988, which was widely regarded as the strongest possible definition.

This paper formally defines fail-stop signatures and shows their relation to ordinary digital signatures. A general construction and actual schemes derived from it follow. They are efficient enough to be used in practice. Next, we prove lower bounds on the efficiency of any fail-stop signature scheme. In particular, we show that the number of secret random bits needed by the signer, the only parameter where the complexity of all our constructions deviates from ordinary digital signatures by more than a small constant factor, cannot be reduced significantly.

Key words. cryptography, authentication, digital signatures, fail-stop, discrete logarithm, factorization, randomization, computational security, information-theoretic security

AMS subject classification. 94A60

PII. S009753979324557X

1. Introduction and overview of results.

Traditional digital signatures, as introduced in [12] and formally defined in [20], allow a person, A (for Alice), to make signatures that everyone who knows A's public key can test. Such signatures are only computationally secure for the signer because they can be forged by persons with sufficiently large computing power. A person able to factor large integers can, for example, very easily forge RSA (Rivest–Shamir–Adleman) signatures (see [38]). Hence the security of these schemes relies on a computational assumption. Moreover, if a signature should be forged, it will be difficult for A to convince the bearer of the signed document or a third party that she did not make that signature.

Fail-stop signatures solve this problem by offering the signer a method for proving that a forgery has taken place. More precisely, even if a forger with unlimited com-

*Received by the editors March 11, 1993; accepted for publication (in revised form) April 3, 1995. This paper is the full version of the extended abstracts [E. van Heyst and T. P. Pedersen, "How to make efficient fail-stop signatures," in *Proc.* 1992 *Eurocrypt*, Lecture Notes in Comput. Sci. 658, Springer-Verlag, Berlin, 1993, pp. 366–377] and [E. van Heijst, T. P. Pedersen, and B. Pfitzmann, "New constructions of fail-stop signatures and lower bounds," in *Proc.* 1992 *Crypto*, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 15–30] together with the definitions of fail-stop signatures, including their relations to ordinary digital signatures. A preliminary version of the definitions was only available in "grey" literature as parts of [B. Pfitzmann, "Für den Unterzeichner unbedingt sichere digitale Signaturen und ihre Anwendung," Diploma thesis, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Karlsruhe, Germany, 1989] and [B. Pfitzmann and M. Waidner, "Formal aspects of fail-stop signatures," Technical Report 22/90, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1990]. Several intermediate papers with previous, less efficient constructions and discussions of applications are only referred to.

http://www.siam.org/journals/

†Cryptomathic, Århus Science Park, DK-8000 Århus, Denmark (tpp@cryptomathic.aau.dk). The research of this author was performed while at the Department of Computer Science of Århus University and supported by the Carlsberg Fondet.

‡Institut für Informatik, Universität Hildesheim, Samelsonplatz 1, D-31141 Hildesheim, Germany, (pfitzb@informatik.uni-hildesheim.de). Future address: Informatik VI, Universität Dortmund, D-44221 Dortmund, Germany. The Isaac Newton Institute in Cambridge, UK hosted this author during the final updates to this paper.

puting power makes an "optimal" forgery, a polynomially bounded signer can prove that the underlying computational assumption has been broken when she sees the forgery (except with negligible probability). Thus the signer can be protected from an arbitrarily powerful forger. Moreover, after the first forgery, all participants, or the system operator, know that the signature scheme has been broken, so that it can be stopped. Hence the name "fail-stop."

1.1. More about ordinary digital signatures. Digital signatures were introduced in [12] and became popular with the RSA scheme [38]. However, it was subsequently discovered that the security requirements made in [12] were too weak, and the structure of signature schemes described there did not allow the desired stronger security. A satisfactory definition was finally published in [20], together with a construction that is secure in this sense (see [20] for the intermediate history). We call such signature schemes "ordinary."

However, the notion of security was always only computational. For signature schemes of the original structure, [12] already showed that this is unavoidable: The signer has a secret key, which she uses to make signatures, and the signatures can be tested by everyone who knows her corresponding public key. Since signing and testing are polynomial-time, one can forge signatures, i.e., find values that pass the test, in nondeterministic polynomial time simply by guessing among all values up to a certain length. Additionally, nonpolynomial lower bounds for problems within NP are not known; hence to prove the security of any ordinary digital signature scheme, one has to make a computational assumption. The same argument also applies to the more general construction in [20]. Hence the effort in the theoretical treatment of signature schemes after [20] concentrated on weakening the necessary assumptions [1, 30, 39].

Note, additionally, that with ordinary digital signatures, it is always the signer whose security relies on the computational assumption. If the signer were allowed to disavow forged signatures, she could also disavow her real signatures (because there is no difference between them), even if the assumption was not broken at all. The recipient's security is unconditional, i.e., all signatures that he has accepted will definitely also be accepted by any third party asked to settle a dispute.

1.2. More about the fail-stop property. The fail-stop property can best be described by considering a judge in a dispute between the signer and a recipient of a digital signature. Usually, the judge will test if the signature is correct and give his verdict—"ok" or "not ok"—accordingly. Fail-stop signatures supply the judge with a third possibility: If the signer can prove that the signature is forged, the judge may say "forgery proved," which can be interpreted as saying that the basic assumption of the system has been broken. Naturally, this possibility of distinguishing forged signatures from authentic signatures only exists as long as the forger has not stolen the signer's key.

The definition of fail-stop signatures does not specify for how much of a system a particular proof of forgery is valid. As long as forging a single signature is provably as hard as breaking a particular computational assumption, it is wise to stop the whole system after any forgery because if one signature could be forged, one must expect that the same forger can make more forgeries. Therefore, the constructions usually assume that there is only one type of proof of forgery. However, it is no problem to make proofs of forgery specific to the keys of individual signers or even (although currently with some loss in efficiency) to each particular signature.

Furthermore, it is not a matter of the definition how one acts after the output "forgery proved." In particular, one obtains exactly the properties of ordinary digital signatures again if the technical verdict "forgery proved" is interpreted just like "signature correct" by the judge and everybody else. On the other hand, if it is agreed that all signatures for which forgery can be proved are rejected, one obtains a signature scheme in which the signer is unconditionally protected against forgeries, whereas the recipient is only computationally protected, i.e., an unrestricted adversary may achieve that a signature accepted by the recipient is later rejected by an honest judge. We call such a signature scheme a "dual signature scheme" because it is dual to ordinary digital signatures with respect to the security for the signer and recipients. Hence fail-stop signatures constitute the first (published) examples of dual signature schemes. Since fail-stop signatures furthermore allow the system to be stopped as soon as the basic assumption has been broken, they are a strictly stronger notion than each of these types of signatures.

As an example where fail-stop signatures, in their special role as dual signatures, may be advantageous, consider an electronic payment system where a customer signs her requests to the bank digitally. Since the bank most likely has much more computing power than the customer and since it can select the system and choose the security parameters, it is reasonable to protect the customer unconditionally, whereas the bank can rely on the customer not having sufficient computing power to repudiate her signatures. Thus the customer should sign with dual signatures and the bank with ordinary digital signatures. For more details about possible benefits of fail-stop signatures and possible advantages for the acceptability of digital signatures in law, see [36, 32].

1.3. Construction idea. Fail-stop signatures work very much like ordinary digital signatures. The signer has a secret key, which she uses to make signatures, and the signatures can be tested by everyone who knows her corresponding public key. A signature that passes this test is called *acceptable*. Now, the basic idea of the existing constructions of fail-stop signatures is that every message has many different acceptable signatures, of which the signer can only construct one (unless she breaks the underlying assumption); this one is called the *correct* signature. However, even an arbitrarily powerful forger does not have sufficient (Shannon) information to determine which of the many acceptable signatures is the correct signature on a new message. Consequently, with very high probability, a forged signature is different from the correct signature. Given a forged signature, the signer can exploit the knowledge of two different signatures on the same message (the forged signature and the correct one) to compute a proof of forgery.

Note that this construction allows an unrestricted signer to disavow her real signatures. However, since this can only occur if the computational assumption has in fact been broken, this is no problem. A proof of forgery does not indicate by *whom* the assumption has been broken, and hence it may have been the signer; but in any case, if this has happened, the system should be stopped.

1.4. Previous results. Fail-stop signatures were first mentioned in [43], and in the reports [31, 35], it was proved that such signatures exist if claw-free pairs of permutations exist; this is a computational assumption known from [20] (see also [4, 36] for descriptions). In particular, this shows that fail-stop signatures exist if factoring large integers or computing discrete logarithms is hard. The construction uses one-time signatures, similar to [25], i.e., messages are basically signed bit by bit.

Therefore, although messages can be hashed before signing and tree authentication is used (similar to [26]), this general construction is not very efficient.

In [32], an efficient variant especially suited for making customers unconditionally secure in on-line payment systems was presented. However, in this scheme, all signatures by one customer with the same key must have the same recipient (e.g., the bank in a payment system), and it is only a dual signature scheme, not a fail-stop signature scheme. Furthermore, signing is a three-round protocol between the signer and the recipient.

1.5. Related types of schemes. In [8], a dual undeniable signature scheme was presented. Undeniable signatures, introduced in [7], are a type of signature in which there is no public predicate that everyone can use to test signatures. Instead, signatures are verified and disavowed using interactive protocols that must be carried out with the signer. This construction was the first signature scheme to achieve unconditional security for signers without bit-by-bit signing. However, it is not as efficient as the following schemes. Although the signatures themselves are efficient, the verification protocol requires quite a lot of computation because it needs σ challenges (similar to signatures) to achieve an error probability of $2^{-\sigma}$. A similar scheme, but with so-called convertible undeniable signatures (cf. [5]), is contained in [22].

In [9], unconditionally secure signatures were introduced, i.e., signature-like schemes where both the signer and the recipients are unconditionally secure. In [37], a transferable version was presented, i.e., signatures can be passed on from one recipient to another. With this extension, unconditionally secure signatures could in principle replace other signatures in many applications. However, these constructions are too inefficient to be used in practice because they require a complicated interactive key-generation protocol and the signatures are very long. We showed in [23] (proofs in [34]) that the latter is unavoidable: to achieve an error probability of $2^{-\sigma}$, the length of unconditionally secure signatures that can be tested by M participants, including those that only have to settle disputes, is at least $M \cdot \sigma$.

1.6. Results in this paper. This paper presents fail-stop signatures in a unified way by giving definitions, efficient constructions, and lower bounds. Preliminary versions of the definitions appeared in [31, 35], and preliminary versions of the constructions and lower bounds appeared in [22, 23].

Definition (see section 3). The formal definition of fail-stop signatures is somewhat more general than that in the report [35] because it allows a more general method for key generation and more memory in the signing algorithm. This increases the scope of validity of the lower bounds.

Constructions (see sections 4 and 5). The first step in our constructions of fail-stop signatures is a general construction based on the concept of bundling homomorphisms, which allows only one message to be signed. This construction has as special cases the two schemes from the extended abstracts [22, 23], which we describe. Signing in the first of these two schemes requires two modular multiplications, whereas signing in the second requires approximately one exponentiation. In both schemes, testing a signature requires approximately two exponentiations. Thus these signatures compare very well with the currently proposed ordinary digital signatures. We then show extensions for signing an arbitrary number N of messages. In special cases, no efficiency is lost; in the general case, the length of signatures and the time needed for testing signatures on short messages grow by a factor of $\log(N)$.

Lower bounds (see section 6). Based on the definition of fail-stop signatures, we prove lower bounds on the size of the keys and the size of fail-stop signatures. For

these bounds, we assume that the probability that a forgery cannot be repudiated is smaller than $2^{-\sigma}$ for some security parameter σ and that the recipients have a similar level of security against very simple brute-force algorithms. Then the most important result is as follows:

- If N messages are to be signed, the signer needs at least $(N+1)(\sigma-1)$ secretly chosen random bits.

Thus the secret key in fail-stop signature schemes is basically a one-time key. However, this does not prevent efficient fail-stop signatures where many messages can be signed: We present an efficient construction where the size of the secret storage space is logarithmic in the number of messages to be signed, and an otherwise less efficient variant where this size is constant. This does not contradict the lower bounds because many secret bits can be deleted soon after they have been generated and used, i.e., the stored secret key varies with time. These constructions with small secret storage are important because secret storage is quite hard to realize since one needs a tamper-resistant device.

Additionally, we show the following:

- The entropy, and hence the length, of a signature is at least $2\sigma - 1$, and the entropy of the public key is at least σ.

These bounds are not much larger than similar bounds for ordinary digital signatures because they concern parameters where the difference between fail-stop signatures and ordinary digital signatures is quite small.

A comparison of the efficient constructions of fail-stop signatures with the lower bounds on unconditionally secure signatures mentioned in section 1.5 shows that fail-stop signatures provide the most viable way of protecting the signer unconditionally.

2. Notation. For a given probability space S (which will always be clear from the context), $\mathrm{P}(E)$ denotes the probability of the event E, and $[S]$ denotes the set of elements with positive probability. Choosing a value from S and assigning it to a variable, x, is denoted by $x \leftarrow S$.

If a is a probabilistic algorithm, $a(i)$ denotes the probability space defined by running a on input i. If $a_1, \ldots, a_n$ are n probabilistic algorithms and p is an n-ary predicate ($n \in \mathbb{N}$), then $\mathrm{P}(p(x_1, \ldots, x_n) :: x_1 \leftarrow a_1(i_1); \ldots; x_n \leftarrow a_n(i_n))$ denotes the probability that the predicate $p(x_1, \ldots, x_n)$ is true after the result of running a_j on input i_j has been assigned to x_j, for $j := 1, 2, \ldots, n$ (in this order). We also allow assignments from other probability spaces, like $x_j \leftarrow S_j$, in such a formula.

The notion "for k sufficiently large" means "$\exists k_0 \forall k \geq k_0$." The characteristic function of a predicate *pred* is denoted by 1_{pred} and the length of a string by $|\cdot|$. The length of a number is the length of the binary representation of that number. This is denoted by $|\cdot|_2$.

The ring of integers modulo a number n is denoted by $\mathbb{Z}_n$, and its multiplicative group, which contains only the integers relatively prime to n, by $\mathbb{Z}_n^*$.

3. Definition of fail-stop signatures. Briefly, a digital signature scheme is defined by three algorithms (see [20]),

1. a key generator,
2. a method for signing, and
3. a method for testing signatures,

such that if the keys are generated correctly using the key generator, then we have the following:

- If the signer signs a message correctly, everyone who knows the signer's public key accepts the signature.

• A polynomially bounded forger cannot make any signature that passes the signature test.

In a fail-stop signature scheme, a protocol is added that allows the (polynomially bounded) signer to prove to third parties that a forged signature is indeed a forgery. It consists of two more algorithms:

4. a method for constructing proofs of forgery, and
5. a method for verifying proofs of forgery (which everyone who knows the public key can carry out).

A proof of forgery is always noninteractive so that it can subsequently be shown to others, and the system can be stopped in consensus. The proof must satisfy two new security requirements:

• The ability to prove forgeries must work independently of the computing power of potential forgers.

• It must be infeasible for the signer to construct signatures that she can later prove to be forgeries.

It can be shown that these two properties imply security against forgery (see section 3.2); hence this security is omitted in the formal definitions.

Until now, nothing has been said about the generation of the secret and public key. In an ordinary digital signature scheme, the primary purpose of the secret key is to enable the signer to make signatures that nobody else can construct, and this key is therefore chosen by either the signer herself or a key-authentication center trusted by the signer. Since it is equally important that fail-stop signatures cannot be forged, the signer still has to take part in choosing the keys. However, since the signer in these schemes is allowed to repudiate (forged) signatures despite the fact that they pass the public signature test, the recipients of signatures must be sure that the signer cannot disavow her own signatures. It is therefore necessary that the recipients or a center trusted by the recipients also participate in the key generation. In particular, such a center is needed if the recipients are not known at the time of choosing the keys. When the signer's public key has been selected, it will usually be stored in a public directory with accepted integrity.

In the following, we first formalize the definition for the case where the signer and a center generate the keys (section 3.1). Section 3.2 shows that every scheme satisfying this definition is secure against forgery, and section 3.3 discusses how the recipients can participate in the key generation.

3.1. Definition. In this section, we consider the situation where the signer and an entity trusted by the recipients generate the keys. This entity is called the *center* and denoted by C.

It should be noted that we use uniform complexity. In particular, the computational assumptions in the next section are described uniformly. It is not difficult to modify the definitions in order to cope with nonuniform complexity, and reductions in the uniform model are automatically valid in the nonuniform model as well.

The security of a cryptographic scheme is usually determined by a security parameter, which specifies the size of the instances of the hard problem used. A fail-stop signature scheme can be broken by either the signer, if she succeeds in constructing a signature that she can later prove to be a forgery, or a forger, if he succeeds in constructing a false signature that the signer cannot repudiate. Since these two attacks have completely different consequences for the participants, it is natural to define the security parameter of a fail-stop signature scheme as a pair (k, σ) of positive integers, where k measures the (computational) security for the recipients and σ determines

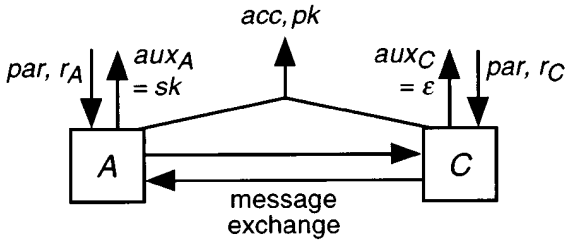

FIG. 1. *Notation used for a correct execution of the key generation G.*

the (unconditional) security for the signer. As is common practice, we shall implicitly assume that these security parameters are represented in unary whenever they are input to an algorithm or protocol.

We allow signing and proving forgeries to be probabilistic and to depend on the history of previously given signatures. Actually, the signatures in our constructions depend only on the *number* of previous signatures. However, the general definition ensures that the lower bounds in section 6 are valid for everything that might reasonably be called a fail-stop signature scheme. We cover all these cases by regarding the random bits as part of the secret key and signing as a deterministic function of the secret key and the sequence of all (previous) messages. This will simplify the notation.

The number of the current message is usually denoted by $i \in \mathbb{N}$.

In addition to σ and k, a fail-stop signature scheme has a parameter $N \in \mathbb{N}$, which is the maximal number of signatures that the signer is willing to construct using the same secret key (hence $1 \leq i \leq N$). As with the security parameters, it is assumed that N is always represented in unary. We often write $par := (k, \sigma, N)$ as an abbreviation of these three parameters.

DEFINITION 3.1. *A* fail-stop signature scheme (*abbreviated* FSS scheme) *with message space* $M \subseteq \{0,1\}^+$ *is a* 5*-tuple* $(G, sign, test, prove, verify)$*, where we have the following:*

- *G is a polynomial-time two-party protocol for generating the keys. The protocol is executed by the signer, A, and C, who both get par as input. Furthermore, each party has a secret random string,* r_A *and* r_C*, respectively. The participants each have a private output channel and there is a (broadcast) channel for common outputs; see Figure* 1*. The common output is* (acc, pk)*, where* $acc \in \{accept, reject\}$ *and pk is the public key (only well defined if* $acc = accept$*). The common broadcast channel can be realized with a usual broadcast channel if one participant outputs pk and the other outputs acc. We generally denote the outputs on the private output channels of the two participants by* aux_A *and* aux_C*, respectively, and we say that G outputs* (acc, pk, aux_A, aux_C)*. If A is executed correctly,* aux_A *is simply the secret key, denoted by sk. If C is executed correctly,* aux_C *is the empty string,* ε*.*

- *sign is a polynomial-time algorithm that on input the secret key, a message number i, and a message sequence* $\underline{m} = (m_1, \ldots, m_i)$ *from M constructs a signature on* m_i*. Thus sign(sk, i,* $\underline{m}$*) denotes the signature on* m_i *if the previously signed messages were* $m_1, \ldots, m_{i-1}$*, and all random bits used are regarded as part of sk (and thus originally of* r_A*). It is called the* correct *signature.*

- *test is a polynomial-time algorithm that on input the public key, a message* $m \in M$*, and a possible signature s on m outputs either ok or notok. If* $test(pk, m, s) = ok$*, we say that s is an* acceptable *signature on m.*

• *prove is a polynomial-time algorithm that on input the secret key, a message* $m \in M$*, a possible*[1] *signature* s *on* m*, and the history hist of previously signed messages* (*plus their signatures*) *either outputs the string "not a forgery" or a bit string proof* $\in \{0,1\}^*$.

• *verify is a polynomial-time algorithm that on input the public key, a message* $m \in M$*, a possible signature* s *on* m *and a string proof outputs either accept or reject. If the result is accept, the string proof is called a valid proof of forgery.*

This 5*-tuple must satisfy the following basic correctness property* (*security is defined separately*)*. For all* $k, \sigma, N \in \mathbb{N}$*, if* A *and* C *follow the computations prescribed by* G*, each output* $(acc, pk, sk, \varepsilon)$ *of* G *satisfies* $acc =$ *accept and for all message numbers* $i \in \{1, \ldots, N\}$ *and message sequences* $\underline{m} = (m_1, \ldots, m_i)$ *from* M*,*

• *if* $s = sign(sk, i, \underline{m})$*, then* $test(pk, m_i, s) = ok$*, i.e., correct signatures are acceptable.*

The output "*not a forgery*" of *prove* signals that *prove* is not able to construct a valid proof of forgery.

Note that *test* and *verify* are not as general as *sign* and *prove* because we have not allowed them to depend on the history of previous signatures: In general, all signatures may go to different recipients. Hence one cannot assume that a recipient (or a verifier) knows anything about the history. Everything he has to know must therefore be included in the signature. One can make an exception only in applications where all signatures of one signer have a fixed recipient or very few recipients.

Before continuing, we introduce some notation to be used with the key-generation protocol. A signer or center who does not necessarily follow the prescribed protocol is denoted by $\tilde{A}$ or $\tilde{C}$, respectively. $G_{\tilde{A},C}$ and $G_{A,\tilde{C}}$ denote the resulting protocols, and $aux_{\tilde{A}}$ and $aux_{\tilde{C}}$ the private outputs. A cheating participant can also use the input par. We denote by $G_{\tilde{A},C}(par)$ the distribution of the outcome of $G_{\tilde{A},C}$ over the random bits r_C of C and $r_{\tilde{A}}$ of $\tilde{A}$ if the common input is par. Similarly, $G_{A,\tilde{C}}(par)$ denotes the distribution of the outcome of $G_{A,\tilde{C}}$.

DEFINITION 3.2. *An FSS scheme is* secure for the recipients *iff the following holds for all probabilistic polynomial-time algorithms* $\tilde{A}$ *and* $\tilde{A}^*$ (*representing the two steps of an attacking signer*)*: If* C *follows the prescribed protocol in an execution of* G *with* $\tilde{A}$ *and if* $\tilde{A}^*$*, on input the private output* $aux_{\tilde{A}}$ *of* $\tilde{A}$*, constructs a triple* $(m, s,$ *proof*)*, then the probability that proof is a valid proof of forgery tends to zero faster than the inverse of any polynomial in* k*. This probability is over the random* (*uniformly distributed*) *choices of* r_C*,* $r_{\tilde{A}}$ *and the random choices used in* $\tilde{A}^*$*.*

More formally, $\forall \tilde{A}$ *and* $\tilde{A}^*$ (*probabilistic polynomial-time*)*,* $\forall \sigma$*,* N*, and* c*, and for* k *sufficiently large,*

$$\mathrm{P}(acc = accept \wedge verify(pk, m, s, proof) = accept :: \\ (acc, pk, aux_{\tilde{A}}, \varepsilon) \leftarrow G_{\tilde{A},C}(par); (m, s, proof) \leftarrow \tilde{A}^*(aux_{\tilde{A}})) < k^{-c}.$$

(*Remember:* $par = (k, \sigma, N)$.)

We now turn to the definition of the security for the signer. Since we have not required that the signer trusts the center, she should be protected no matter what the center does in the key-generation protocol as long as she follows the protocol herself. Furthermore, the purpose of fail-stop signatures is that the signer should be secure

[1]The algorithm is primarily intended to be used on acceptable signatures. However, it is more convenient for the notation to define it more generally. If s is not an acceptable signature, the output will usually be "*not a forgery.*"

even against arbitrarily powerful forgers. We therefore consider arbitrarily powerful centers $\tilde{C}$. Remember that $[G_{A,\tilde{C}}(par)]$ denotes the set of possible outcomes of the protocol if *par* is given.

DEFINITION 3.3. *Let an FSS scheme with message space $M \subseteq \{0,1\}^+$ and parameters $par = (k, \sigma, N)$ be given, and consider an arbitrarily powerful center $\tilde{C}$. If A and $\tilde{C}$ have executed G and the outcome was $(acc, pk, sk, aux_{\tilde{C}})$ with $acc = accept$, we define the following:*

(a) *The set of possible* histories *is*

$$Hist(sk) := \{((m_1, \ldots, m_j), (s_1, \ldots, s_j)) | 1 \le j \le N \\ \wedge\ (m_i \in M \wedge s_i = sign(sk, i, (m_1, \ldots, m_i)) \text{ for } i = 1, \ldots, j)\}.$$

For a given history hist, let $M(hist)$ denote the set of sign messages in hist.

(b) *The set of* possible secret keys (*from the point of view of an unrestricted forger*) *given $pk, aux_{\tilde{C}}$, and a history hist is*

$$SK_{\tilde{C}}(pk, hist, aux_{\tilde{C}}) \\ := \{sk | (accept, pk, sk, aux_{\tilde{C}}) \in [G_{A,\tilde{C}}(par)] \wedge hist \in Hist(sk)\}.$$

$SK_{\tilde{C}}(pk, hist, aux_{\tilde{C}})$ is equipped with a distribution induced by the random bits of the signer and corresponds to the remaining uncertainty that an attacker has about the real secret key when he tries to choose an "optimal" forgery.

(c) *The set of* successful forgeries *given pk and a history hist is*

$$Forg(pk, hist) := \{f = (m, s) | m \in M \backslash M(hist) \wedge test(pk, m, s) = ok\},$$

i.e., the set of acceptable signatures on messages not contained in the history.

(d) *A value $f = (m, s) \in Forg(pk, hist)$ is a* provable forgery *after a history hist, abbreviated provable$(sk, pk, hist, f)$, iff*

$$verify(pk, m, s, prove(sk, m, s, hist)) = accept,$$

i.e., if applying prove to it yields a valid proof of forgery.

Intuitively, the following definition says that no matter what an arbitrarily powerful center does during the key generation, no matter what messages A signs, and no matter what message and signature the forger selects as a forgery given all this knowledge, A can repudiate the forged signature with high probability. The probability is over the possible secret keys (among which, roughly speaking, the forger must guess which one A really has). For the first similar definition of security against an unrestricted adversary, see [42].

There are two possible sources for a small error probability: One is during key generation, where A might be tricked into accepting a bad key pair; the other is that a forger happens to find exactly A's correct signature.

DEFINITION 3.4. *Let an FSS scheme be given.*

(a) *For any given parameters $k, \sigma, N \in \mathbb{N}$ and any arbitrarily powerful center $\tilde{C}$, define a set $Good_{\tilde{C}}$ of "good" outcomes of the key generation as follows:*

$$(sk, pk, aux_{\tilde{C}}) \in Good_{\tilde{C}} :\Leftrightarrow \forall hist \in Hist(sk), \forall f \in Forg(pk, hist):$$

$$\mathrm{P}(provable(sk', pk, hist, f) :: sk' \leftarrow SK_{\tilde{C}}(pk, hist, aux_{\tilde{C}})) \ge 1 - 2^{-\sigma}.$$

Thus an outcome is called good if it guarantees that after any history, an unrestricted forger still has so much uncertainty about the secret key that his forgery will be provable with very high probability.

(b) *The FSS scheme is* secure for the signer *iff* $\forall\sigma$, $\forall par = (k, \sigma, N)$, *and for any arbitrarily powerful center* $\tilde{C}$,

$$\mathrm{P}((sk, pk, aux_{\tilde{C}}) \notin Good_{\tilde{C}} \wedge acc = accept :: (acc, pk, sk, aux_{\tilde{C}}) \leftarrow G_{A,\tilde{C}}(par)) \leq 2^{-\sigma}.$$

(c) *If both A and C follow G, each output is good, i.e.,* $(sk, pk, \varepsilon) \in Good_C$.

DEFINITION 3.5. *A* secure FSS scheme *is an FSS scheme that is secure for both the signer and the recipients.*

3.2. Relation to ordinary digital signatures: Security against forgery. In the above definition of secure fail-stop signatures, we have not explicitly demanded that it be difficult for a forger to construct acceptable signatures. Clearly, a signature scheme is useless if it is easy to make forgeries, even if they can be repudiated. We now show that Definition 3.5 actually implies that forging is hard for polynomially bounded enemies. More precisely, we show that existential forgery is infeasible even after an adaptive chosen-message attack (see [20]).

In an adaptive chosen-message attack against a signature scheme with security parameter l (which is used as an input in key generation) a forger F does the following, given the public key as an input:

1. Repeat a polynomial number of times (in l): Generate (in some way) a message m and receive the correct signature on m from the signer.
2. Output a pair (m', s'). (This should be a new message–signature pair.)

DEFINITION 3.6. *A signature scheme with security parameter* l *is* secure against an adaptive chosen-message attack *iff for all* $c > 0$ *and for all probabilistic polynomial-time forgers* F *as above, the probability that* F *outputs a pair* (m', s') *such that* m' *is different from all messages chosen in step* 1 *and* s' *is an acceptable signature on* m' *is less than* l^{-c} *for* l *sufficiently large.*

The probability is over the random bits used in the key generation, the random bits of F, and the random choices of the signatures, if the signing algorithm is probabilistic.

We now consider the security of fail-stop signatures against a forger who has not participated in the key generation.

THEOREM 3.1. *A secure FSS scheme is secure against an adaptive chosen-message attack by forgers who do not participate in G, i.e., who have only* par *and the result pk of a correct execution of G as inputs; the parameter k of the FSS scheme plays the role of the parameter l in Definition* 3.6.

Proof. Let a secure FSS scheme be given as in Definitions 3.1 and 3.5. Assume further that there exists a probabilistic polynomial-time forger F as above that on input (par, pk) outputs a pair of a new message and an acceptable signature (m', s') with probability $p(par)$. This probability is over all the random bits used in key generation and in F. Since F runs in polynomial time, a signer who tries to cheat a recipient can use the same algorithm:

1. Execute G correctly with C. This yields a key pair (sk, pk).
2. Execute F alternating with the real signing algorithm to obtain a history *hist* and a pair (m', s').
3. Use *prove* to compute a proof of forgery for (m', s'), given sk and *hist*.

Roughly, the argument is as follows: On one hand, the security for the signer against a forger with algorithm F implies that step 3 leads to a valid proof of forgery

with nonnegligible probability. On the other hand, this means that when the signer alone carries out the whole algorithm (i.e., step 1 as $\tilde{A}$ and steps 2 and 3 as $\tilde{A}^*$), it contradicts the security for the recipients.

We now proceed more formally. The history and message–signature pair generated in step 2 have exactly the same distribution as those that the forger would have constructed (i.e., the distribution used in Definition 3.6). This implies two things: First, (m', s') is a successful forgery with probability $p(par)$. Second, whenever this is the case, the signer can repudiate the forgery, i.e., step 3 yields a valid proof of forgery with probability at least $q := 1 - 2^{-\sigma}$, because $(sk, pk, \varepsilon) \in \mathit{Good}$ (see Definition 3.4(a, c)). Here q is the a posteriori probability over the possible secret keys, given any history. We omit the tedious details needed to combine these different types of probabilities formally before one obtains the expected result:

With probability at least $p(par)(1 - 2^{-\sigma}) \geq p(par)/2$, a signer executing steps 1, 2, and 3 obtains an acceptable signature together with a valid proof of forgery for it. Since the FSS scheme is secure for the recipients, this implies that $p(par)$ must be negligible as a function of k. This proves the theorem. □

This theorem can be interpreted as saying that fail-stop signatures are a stronger notion of signatures than that defined in [20] in two ways. The first way is described in the following theorem.

THEOREM 3.2. *Every secure fail-stop signature scheme can be used to construct an (equally efficient) secure digital signature scheme in the sense of* [20].

Proof. The main reason that a fail-stop signature scheme itself does not fulfill the definition in [20] is the interactive key generation. However, one can construct such a scheme as follows: On input (k, N), the signer executes both A and C in the key generation with $\sigma := 1$. The algorithms *sign* and *test* remain the same, and *prove* and *verify* are omitted. Theorem 3.1 implies that this is a secure signature scheme in the sense of [20]. □

As the second way, section 3.3 shows how Theorem 3.1 can be strengthened for the fail-stop signature scheme itself so that unconditional security for the signer is combined with security against an adaptive chosen-message attack by anyone, i.e., without even trusting a center.

3.3. Fail-stop signatures with known recipients. In the definitions in section 3.1, the keys were generated by the signer and a center trusted by the recipients. This section briefly discusses how fail-stop signature schemes work if the recipients themselves participate in the key generation. More precisely, one should distinguish recipients and risk bearers. Risk bearers are those who have a disadvantage if a proof of forgery is accepted and thus a signature becomes invalid. For instance, a recipient might have an insurance that covers his losses if signatures that he had accepted are proved to be forgeries. Then the insurance company—and not the recipient—is the risk bearer and has to trust the key generation process. We continue to say recipient for simplicity.

One recipient. In this case, we can simply replace the trusted center by the recipient. Note that a fail-stop signature scheme with only one recipient is not useless because everyone who knows the public key can still test the signatures and verify the proofs of forgery. The only restriction is that only the intended recipient is guaranteed that the signer cannot repudiate her own signatures. Such a scheme can, e.g., be applied in electronic payment systems for signing the customers' requests to the bank.

Many recipients. The definition of FSS schemes with many known recipients also follows the previous definitions very closely. In such a scheme, all the recipients participate in a key-generation protocol with the signer. Each participant has the input *par* and a secret random string, and the common output is either *reject* or $(accept, pk)$. (We assume that all recipients are satisfied with the same parameter k for their computational security.) The signer gets the secret key corresponding to pk as private output. The security for the signer can be defined as before, and the security for the recipients is essentially defined by requiring Definition 3.2 for each of them. In other words, if a recipient follows the key-generation protocol correctly, he is assured that a (polynomially bounded) signer cannot repudiate her own signatures, even if the signer and the remaining recipients cooperate.

We now show how FSS schemes with many known recipients can be constructed from FSS schemes in the sense of section 3.1. In some FSS schemes, the only task of the center in the key-generation protocol is to select a random string. In these schemes, it is quite easy to replace the center by many recipients because they only have to perform a multiparty coin-flipping protocol.

In the general case, one can apply a protocol for general secure multiparty computations in a certain way. For some details, see [35, 34]. However, in the present state of cryptography, the following simpler method is much more efficient. Its disadvantage is that the keys are long and thus signing and testing are relatively inefficient.

Construction 3.1 (many keys). Let R be the number of recipients. Each recipient executes protocol G with the signer once. This results in R key pairs, $(sk_j, pk_j)_{j=1,\ldots,R}$. The secret key of the signer is defined as $(sk_1, \ldots, sk_R)$ and the public key as $(pk_1, \ldots, pk_R)$. All signatures as well as proofs of forgery consist of R parts, one for each of the R key pairs. Note that everyone can test all parts of each signature or proof of forgery. ◇

The security for the jth recipient is guaranteed because if he carries out his execution of G correctly, it is computationally infeasible to compute valid proofs of forgery for pk_j. The security for the signer is guaranteed because given a successful forgery, she can compute proofs of forgery for it for each key pair with high probability.

As an immediate consequence of Theorem 3.1, this scheme is secure against an adaptive chosen-message attack if at least one of the recipients executes G correctly with the signer—even if the forger cooperates with the remaining $R - 1$ recipients. We can even achieve security if the forger may cooperate with all recipients by letting the signer take part in the key generation in the role of a recipient as well, i.e., she generates an additional key pair (sk_{R+1}, pk_{R+1}) all on her own, which is then treated like any other key pair.

4. Constructions of fail-stop signature schemes. This section first presents a general construction of a fail-stop signature scheme, which is subsequently used in actual instantiations. We describe two such instantiations based on the assumptions that it is hard to compute discrete logarithms and to factor integers, respectively. These are the two best-known concrete computational assumptions used in cryptography, and they are therefore well investigated and fairly trustworthy.

The constructions as described in this section allow only one message to be signed; hence they are called one-time signature schemes. Section 5 presents extensions for signing an arbitrary number of messages.

4.1. Bundling homomorphisms. In order to present our general construction of fail-stop signatures, we first define bundling homomorphisms. They are a special type of cryptographic hash functions and may have other uses in cryptography. Briefly,

a bundling homomorphism h is a homomorphism $h : G \to H$ between two Abelian groups $(G, +, 0)$ and $(H, \cdot, 1)$ that satisfies the following:

- Every image $h(x)$ has at least 2^τ preimages. (The bundling homomorphism is said to be of degree 2^τ.)
- It is infeasible to find collisions, i.e., two different elements that are mapped to the same value by h.

In order to make the second of these requirements precise, we have to consider a family of such functions. The individual functions of this family are indexed by a key K. The key is chosen depending on two security parameters: τ, which determines the bundling degree, and k, which measures the computational security against collision-finding. The parameters τ and k are part of the index to this function, but we only write K instead of (K, τ, k). This leads to the following definition.

DEFINITION 4.1. *A* family of bundling homomorphisms *is a quadruple* $(g, h, \mathcal{G}, \mathcal{H})$, *where we have the following:*

- g, *the key generator, is a probabilistic polynomial-time algorithm that on input parameters* $k, \tau \in \mathbb{N}$ *outputs a value* K. *Let* $\mathcal{K}$ *be the set of all possible keys, i.e., the union of the sets* $[g(k, \tau)]$ *for all* $k, \tau \in \mathbb{N}$.
- $\mathcal{G}$ *and* $\mathcal{H}$ *are families of Abelian groups, one for each key. More formally,* $\mathcal{G} = (G_K, +, 0)_{K \in \mathcal{K}}$ *and* $\mathcal{H} = (H_K, \cdot, 1)_{K \in \mathcal{K}}$.
- h *is a polynomial-time algorithm that on input* $K \in \mathcal{K}$ *and* $x \in G_K$ *outputs a value* $z \in H_K$. *The restriction of* h *to a particular key* K *is abbreviated as* h_K.

This quadruple must satisfy the following properties:

(a) *Each* h_K *is a group homomorphism from* $(G_K, +, 0)$ *to* $(H_K, \cdot, 1)$.

(b) *For all* $k, \tau \in \mathbb{N}$, $K \in [g(k, \tau)]$, *each* $z \in h_K(G_K)$ *has at least* 2^τ *preimages under* h_K.

(c) *The family is collision-resistant: For every* $c > 0$ *and for every probabilistic polynomial-time algorithm* $\tilde{A}$, *the probability that* $\tilde{A}$ *on input* $K \in [g(k, \tau)]$ *outputs a pair* (x, x') *such that* $x \neq x'$ *and* $h_K(x) = h_K(x')$ *is less than* k^{-c} *for* k *sufficiently large. More precisely,* $\forall \tau \forall c \, \exists k_0 \forall k \geq k_0$,

$$\mathrm{P}(h_K(x) = h_K(x') \wedge x \neq x' :: K \leftarrow g(k, \tau); (x, x') \leftarrow \tilde{A}(K)) < k^{-c}.$$

(*A more common name for collision-resistant is collision-free, but collision-resistant emphasizes the computational aspect better.*)

Additionally, there must be polynomial-time algorithms (*which do not need explicit names in the following*) *that on input* K

- *compute the operations in the two groups* $(G_K, +, 0)$ *and* $(H_K, \cdot, 1)$,
- *select elements of* G_K *uniformly at random, and*
- *test membership in* H_K *and* G_K.

Note that k_0 depends on τ in the definition of collision resistance. Actually, all of our constructions satisfy the stronger requirement where k_0 is independent of τ, as long as τ is polynomial in k (because an arbitrarily long input τ would give the collision finder time more than polynomial in k).

4.2. The general construction. The general construction yields a rather special case of the definition of fail-stop signature schemes. First, only one message is signed. Second, the key generation of this scheme (like all previous constructions in the literature) is quite simple. These two properties are now defined formally because the constructions to sign many messages in section 5 can be based on any FSS scheme with these properties.

DEFINITION 4.2. *A* one-time fail-stop signature scheme with prekey *is defined like a general FSS scheme in Definition* 3.1*, except that the parameter* N *is* 1 *and the key-generation protocol* G *is of a special, simpler form: It is constructed from a triple* $(gen_C, (P,V), gen_A)$*, where we have the following:*

- gen_C *is a probabilistic polynomial-time algorithm that on input par generates values prek (the* prekey*) and* w *(called witness).*
- (P,V)*, the prekey verification protocol, is a polynomial-time two-party protocol where* P *gets the input* $(par, prek, w)$ *and* V *only gets* $(par, prek)$*. As a result,* V *outputs accept or reject. Correct outputs of the prekey generation should always be accepted, i.e., if* $(prek, w) \in [gen_C(par)]$*, the output of* V *should be accept. The most efficient special case is where* P *does not do anything, i.e.,* V *decides locally whether or not to accept prek.*
- gen_A *is a probabilistic polynomial-time algorithm that on input a prekey prek outputs a key pair* (sk, pk)*. It is called the main key-generation algorithm.*

The protocol G *is constructed from these subprotocols as follows: First, the center* C *executes* gen_C *and publishes the resulting prekey. Next,* (P,V) *is executed for this prekey by the center* (P) *and the signer* (V)*, where the center has* w *as an additional input. (The purpose of this step is to prove some desired property of prek to the signer.) Finally, the signer* A *carries out* gen_A *on input prek. We say that she generates a key pair based on the prekey prek. The public key in the sense of Definition* 3.1 *is the pair* $(prek, pk)$*. However, we often omit prek in the notation because it is clear from the context.*

The signer, using V, may also accept if *prek* is not a possible outcome of the correct prekey generation; we only use (P,V) to prove those properties of *prek* to the signer that are necessary for the signer's security. This is often much more efficient than proving correct generation. Note that (P,V) is similar to an interactive proof system [19], but our prover is polynomial-time and needs the witness w.

Schemes with prekey have the following advantage if there are several signers. The center can publish the prekey without knowing which signers will take part. Every signer carries out the interactive proof with the center once and can then base many successive secret keys on this prekey without further interaction with the center. This is exploited in section 5. The security is not weakened if many signers base their key pairs on the same prekey because of the following:

- If this led to signers being able to repudiate their own signatures, one cheating signer alone could generate many key pairs based on that prekey and experiment with them locally until she could repudiate a signature, and only then would she publish the corresponding public key.
- If this allowed forgers to make unprovable forgeries with nonnegligible probability (over the choice of the key pairs of all the signers), the probability for each key pair would also be nonnegligible because all key pairs are identically distributed.

We now describe a framework for constructing a one-time FSS scheme with prekey from a family of bundling homomorphisms. A few parameters are left open; they depend on the choice of the family of bundling homomorphisms. We then present two theorems that reduce the security of the scheme to one property of the bundling homomorphisms and those parameters. This property will be proved and the parameters specified in the instantiations in sections 4.3 and 4.4.

Since the message number i is always 1, it will be omitted in the notation, and instead of a message sequence $\underline{m}$ of length 1, a message m is written.

Construction 4.1 (general construction). Let a family of bundling homomor-

phisms with a key generator g be given. Then the various components of a one-time fail-stop signature scheme with prekey are defined as follows:

- Key generation: Let security parameters k and σ be given. The parameter τ for the degree of the bundling homomorphisms is a function of σ, which will be specified later.
- Prekey generation gen_C: The center computes $K \leftarrow g(k, \tau)$ and publishes it, i.e., $prek := K$. This corresponds to choosing one homomorphism h_K of the family. Henceforth, let $h := h_K$, $G := G_K$, and $H := H_K$. Instead of g, an algorithm g' may be used that outputs K with the same probability distribution as g but also outputs a witness w.
- Prekey verification (P, V): The signer must be assured that K is a possible outcome of $g(k, \tau)$, or at least that h is a homomorphism and satisfies Definition 4.1(b). The choice of a predicate between these two extremes that can be proved most efficiently depends on the choice of the family of bundling homomorphisms. A prekey K is called *good* if it fulfills this predicate; otherwise, it is called *bad*. If possible, this protocol is a local test by the signer. Otherwise, it is a zero-knowledge proof from the center to the signer (see [19, 18]—except we require P to be polynomial-time and to use a witness w). The probability that the signer accepts the proof for a bad prekey K must be at most $2^{-\sigma}$ for each K.[2]
- Main key generation gen_A: The signer generates her secret key $sk := (sk_1, sk_2)$ by choosing sk_1 and sk_2 randomly in G, and she computes her public key $pk := (pk_1, pk_2)$, where $pk_i := h(sk_i)$ for $i = 1, 2$.
- The message space M is a subset of $\mathbb{Z}$ depending on the choice of the prekey.
- Signing: The correct signature on a message m in the message space is

$$sign(sk, m) := sk_1 + m\, sk_2.$$

(Multiplication with elements of $\mathbb{Z}$ is, as usual, defined by repeated addition.)

- Test: The algorithm *test*, which determines whether a signature $s \in G$ is acceptable, is defined so that $test(pk, m, s) = ok$ iff $pk_1 \cdot pk_2^m = h(s)$.
- Proof of forgery: Given an acceptable signature $s' \in G$ on m such that $s' \neq sign(sk, m)$, the signer computes $s := sign(sk, m)$ and $proof := (s, s')$.
- Verification of proof of forgery: Given a pair (x, x'), verify that x and x' are elements of G, that $x \neq x'$, and that $h(x) = h(x')$. $\diamond$

This concludes the description of the general construction. The following theorem shows that any instantiation of this construction works and that it is secure for the recipients.

THEOREM 4.1. *For any family of bundling homomorphisms, and with any choice of the parameters that have been left open, the general construction has the following properties:*

(a) *Correct signatures pass the test* (*i.e., Definition* 3.1 *is fulfilled*).

(b) *A polynomially bounded signer cannot construct a signature and a valid proof that it is a forgery* (*i.e., Definition* 3.2 *is fulfilled*).

(c) *If s^* is an acceptable signature on m^* and $s^* \neq sign(sk, m^*)$, the signer obtains a valid proof of forgery* (*this is a step towards fulfilling Definition* 3.4).

(d) *If all values $(sk, (K, pk), aux_{\tilde{C}})$ where K is good are contained in $Good_{\tilde{C}}$, the scheme is secure for the signer* (*another step towards fulfilling Definition* 3.4).

[2]Note that the definition of zero-knowledge proofs from [19] only requires that the probability that a bad value is accepted decreases faster than the inverse of any polynomial, but the actual proofs in [18] have the strictly exponential decrease that we need.

Proof. Part (a) follows from the fact that h is a homomorphism:

$$h(s) = h(sk_1 + m\,sk_2) = h(sk_1) \cdot h(sk_2)^m = pk_1 \cdot pk_2^m.$$

Part (c) is a trivial consequence of the fact that both s and s^* pass the test. Part (d) follows immediately from the requirement that we have made on the error probability of the prekey verification. For part (b), note that a proof of forgery is exactly a collision of the bundling homomorphism h_K, where K is chosen by the center in gen_C with the correct probability distribution. Hence part (b) follows immediately from the collision resistance of the bundling homomorphisms, except that we have to show that the zero-knowledge proof does not make it easier for the signer to find collisions, which is intuitively clear (although a formal proof is lengthy; see [34]). □

This theorem shows that the general construction is secure for the recipients and that it is also secure for the signer if an arbitrarily powerful forger cannot guess a correct signature $sign(sk, m^*)$, except with a very small probability, whenever the prekey is good. In order to estimate the probability with which a forger can find such a signature, we first note that given a public key, at least $2^{2\tau}$ secret keys are possible. Given a correct signature on another message m, the forger has more information about sk, but Theorem 4.2 gives a condition under which this information is not sufficient to guess the correct signature on m^* with too high a probability.

THEOREM 4.2. *Consider Construction* 4.1. *Let parameters k and σ, a good prekey K, and two messages $m \neq m^*$ from the corresponding message space be given. Let*

$$T := \{d \in G | h(d) = 1 \wedge (m^* - m)d = 0\}.$$

Then for all key pairs $(sk, pk) \in [gen_A(K)]$ and all values $s^* \in G$ (a forgery) satisfying $test(pk, m^*, s^*) = ok$, the probability that $s^* = sign(sk, m^*)$, given $s := sign(sk, m)$, is at most $|T|/2^\tau$, i.e., for any center $\tilde{C}$,

$$\mathrm{P}(s^* = sign(sk', m^*) :: sk' \leftarrow SK_{\tilde{C}}(pk, (m, s), aux_{\tilde{C}})) \leq |T|/2^\tau.$$

Note that the probability in this theorem resembles Definition 3.4(a).

Proof. Since K is good, h is at least a homomorphism and satisfies Definition 4.1(b). Note that $aux_{\tilde{C}}$ can contain additional information about K but that the only information that the signer divulges about sk is pk and a correct signature s on one message m. The set of possible keys given this information is

$$\begin{aligned} SK_{\tilde{C}} &= \{(sk'_1, sk'_2) \in G \times G | h(sk'_1) = pk_1 \wedge h(sk'_2) = pk_2 \wedge sk'_1 + m\,sk'_2 = s\} \\ &= \{(s - m\,sk'_2, sk'_2) | h(sk'_2) = pk_2\} \end{aligned}$$

because h is a homomorphism and $s = sk_1 + m\,sk_2$. The size of $SK_{\tilde{C}}$ is therefore at least 2^τ. We must now find out how many of these keys satisfy $s^* = sign(sk', m^*)$, i.e.,

$$(*) \qquad sk'_1 + m^*\,sk'_2 = s^*.$$

Since we only consider keys in $SK_{\tilde{C}}$, we can replace sk'_1 by $s - m\,sk'_2$. Hence $(*)$ is equivalent to

$$(m^* - m)sk'_2 = s^* - s.$$

This equation might be unsolvable, but if there is any solution sk_2'', the set of all solutions in $SK_{\tilde{C}}$ is

$$\{(s - m\,sk_2', sk_2') | h(sk_2') = h(sk_2'') \wedge (m^* - m)(sk_2' - sk_2'') = 0\}.$$

Hence the number of solutions is $|T|$ (where d corresponds to the difference $sk_2' - sk_2''$). Since sk is uniformly distributed in Construction 4.1, all elements of $SK_{\tilde{C}}$ are equally probable, and the attacker is successful with probability at most $|T|/|SK_{\tilde{C}}| \leq |T|/2^\tau$, as claimed. □

COROLLARY. *Theorem* 4.2, *together with Theorem* 4.1(c, d), *shows that the general construction is secure for the signer* (*as in Definition* 3.4) *if* τ *is chosen so that* $|T|/2^\tau \leq 2^{-\sigma}$, *i.e.,* $\tau \geq \sigma \log_2(|T|)$.

Consequently, we must find the maximal size of

$$T_{m'} := \{d \in G | h(d) = 1 \wedge \mathrm{ord}_G(d) | m'\}$$

over all possible differences m' of two messages. The size of this set depends on the chosen family of bundling homomorphisms.

Note that the collision resistance of h was only needed in Theorem 4.1(b), i.e., to ensure the security for the recipients. This is the reason why the center need not prove to the signer that h has this property.

4.3. A scheme based on discrete logarithms. We now construct a family of bundling homomorphisms for which finding collisions is equivalent to computing discrete logarithms.

Let p and q be large primes such that q divides $p - 1$, and let H_q be the unique subgroup of $\mathbb{Z}_p^*$ of order q. (Recall the theorem that all groups $\mathbb{Z}_p^*$ are cyclic and of order $p - 1$.) Remember that all elements of H_q except 1 are generators. We shall assume that it is hard to compute discrete logarithms in H_q (as already sketched in [12]). For elements a and b of H_q, where $a \neq 1$, the discrete logarithm $\log_a(b)$ is defined as the number $e \in \{0, \ldots, q - 1\}$ with $a^e = b$.

Assumption DL. For all probabilistic polynomial-time algorithms D, for all $c \in \mathbb{N}$, and for k sufficiently large, the probability that D, on input two primes p and q, where q is a k-bit prime dividing $p - 1$ and $q > (p - 1)/q$, and two generators a and b of H_q, outputs $\log_a(b)$ is less than k^{-c}. The probability is over the random bits used by D and the uniform random choices of p, q, a, and b with the given constraints. ◇

The scheme presented below actually works for any groups of prime order, but for the sake of concreteness, we shall assume the above setup.

Construction 4.2 (discrete logarithm homomorphisms).

- Key generator g: On input k and τ, it chooses primes p and q as above with $|q|_2 = \max(k, \tau)$ and two random generators a and b of H_q. It outputs the key $K := (p, q, a, b)$.
- Families of groups: Define

$$G_q := \mathbb{Z}_q \times \mathbb{Z}_q$$

with pairwise addition (modulo q). With a slight misuse of the notation, we abbreviate $G_{(p,q,a,b)} =: G_q$ and $H_{(p,q,a,b)} =: H_q$.

- The homomorphisms are defined as

$$h_{(p,q,a,b)}\colon G_q \to H_q; \qquad h_{(p,q,a,b)}(x, y) := a^x b^y.$$

The corresponding algorithm h is clear. ◇

Similar constructions were first used in [6].

THEOREM 4.3. *Under Assumption* DL*, Construction* 4.2 *is a family of bundling homomorphisms.*

Proof. It is easy to see that all necessary operations are efficiently computable. In particular, g generates q first and then searches for a prime p among the numbers of the form $tq+1$. Now the properties from Definition 4.1(a–c) must be verified.

(a) Obviously, each $h_{(p,q,a,b)}$ is a group homomorphism. (Recall that the order of H_q is q.)

(b) For every $z \in H_q$, there are exactly q elements (x, y) of G_q that h maps to z: For each x, there is exactly one y with $b^y = za^{-x}$ because b is a generator.

(c) Assume that a probabilistic polynomial-time algorithm $\tilde{A}$ could compute collisions of h with nonnegligible probability. We then construct an algorithm D that on input (p, q, a, b) computes the discrete logarithm of b with respect to a as follows: First, D runs $\tilde{A}$, and if $\tilde{A}$ outputs a collision, i.e., $(x, y) \neq (x', y')$ with $a^x b^y = a^{x'} b^{y'}$, then D computes $\log_a(b)$ as $(x'-x)(y-y')^{-1} \bmod q$. (Note that $y = y'$ is impossible.) D is successful with the same probability as $\tilde{A}$ and almost equally efficient. Hence it contradicts Assumption DL. □

This theorem implies that we can construct an FSS scheme that is secure for the recipients under the discrete logarithm assumption. Before we go into the details of the security for the signer, the resulting scheme is described, and the parameters that were left open in the general construction are fixed.

Construction 4.3 (the Discrete Logarithm Scheme).

- Key generation: On input k and σ, the parameter τ for the degree of the bundling homomorphisms is set to σ.
- Prekey generation: The center selects primes p and q and generators a and b (with the algorithm g) and publishes them.
- Prekey verification: The signer can verify by herself that p and q are primes and that a and b are generators of H_q by verifying that their order is q. Hence key generation is extremely simple in this scheme.
- Main key generation: The secret key consists of four numbers x_1, y_1, x_2, and y_2 between 0 and $q-1$ (more precisely $(x_1, y_1), (x_2, y_2) \in G_q$), and the corresponding public key is the pair (pk_1, pk_2) with

$$pk_1 := a^{x_1} b^{y_1} \quad \text{and} \quad pk_2 := a^{x_2} b^{y_2}.$$

- The message space is defined as $\{0, 1, \ldots, q-1\}$.
- Signing: The correct signature on a message m from this space is

$$(x, y) := (x_1, y_1) + m(x_2, y_2) = (x_1 + m\ x_2, y_1 + m\ y_2).$$

- Test: A pair (x, y) is an acceptable signature on the message m iff

$$a^x b^y = pk_1 pk_2^m.$$

- Proofs of forgery and their verification: According to the general construction, a proof of forgery is a collision of h. Hence such a proof consists of four numbers. However, the signer can just as well show $e := \log_a(b)$ as a proof of forgery because this is equivalent to the other proof but shorter and easier to verify. (e yields a collision $a^e b^0 = a^0 b^1$.)

We now return to proving that this scheme is also secure for the signer.

THEOREM 4.4. *The Discrete Logarithm Scheme is secure for the signer.*

Proof. According to the end of section 4.2, we have to find the maximal size of the set

$$T_{m'} = \{d \in G_q | h(d) = 1 \wedge \mathrm{ord}(d) | m'\}$$

for all values of m' between 1 and $q-1$ (m' is the difference between two different legal messages; negative values need not be considered separately), given that the prekey is good. Thus q is prime, and the order of all nonzero elements of G_q is q. Hence $(0,0)$ is the only element of $T_{m'}$.

Together with the corollary to Theorem 4.2, this implies that it suffices to choose $\tau := \sigma$ in the proposed scheme, as we did in Construction 4.3. □

The choice of $\tau := \sigma$ means that $|q|_2$ is chosen as $\max(k, \sigma)$. For reasonable parameters k and σ, this means $|q|_2 = k$.

We conclude this section with a short evaluation of the efficiency of the proposed scheme:

- Signing requires two multiplications modulo q.
- Testing a signature requires less than two exponentiations modulo p. This is because the recipient can test the signature (x, y) by computing $a^x b^y\, pk_2^{-m}$ and verifying that it equals pk_1. This corresponds to between k and $2k$ modular multiplications, depending on the message, with a suitable exponentiation algorithm.
- The length of the secret key (for one message) is $4k$.
- The length of the public key is $2|p|_2$ bits.
- The length of a signature on a k-bit message is $2k$.

4.4. A scheme based on factoring. This section shows how the general construction can be used to construct fail-stop signatures where the security for the recipients relies on the intractability of factoring. The family of bundling homomorphisms was defined in [4], using ideas from [20, 17]. A similar homomorphism was already used in [2]. We denote the group of quadratic residues modulo an integer n by $QR_n := \{x \in \mathbb{Z}_n^* \,|\, \exists w\colon\ w^2 \equiv x \bmod n\}$. The basic assumption of this scheme is that it is hard to factor large integers.

Assumption F (see [20]). For all probabilistic polynomial-time algorithms F, for all $c \in \mathbb{N}$, and for k sufficiently large, the probability that F, on input a k-bit integer n which is the product of two primes p and q of equal length and with $p \equiv 3$ and $q \equiv 7 \bmod 8$, outputs one of the factors is less than k^{-c}. The probability is over the random bits used by F and the uniform random choice of n with the given constraints. ◇

Construction 4.4 (factoring homomorphisms).

- Key generator g: On input k and τ, it chooses a k-bit integer $n = pq$ such that p and q are primes with $p \equiv 3$ and $q \equiv 7 \bmod 8$. It outputs $K := (\tau, n)$.
- Families of groups: We slightly abbreviate the groups for a key $K = (\tau, n)$ as

$$H_n := (\pm QR_n)/\{1, -1\} \quad \text{and} \quad G_{\tau,n} := \mathbb{Z}_{2^\tau} \times H_n.$$

The group operation in H_n is induced by multiplication modulo n. Each element of H_n, which is a coset $\{x, -x\}$, will be identified with its smaller member, i.e., a number between 0 and $n/2$. The reason for using the factor group H_n instead of QR_n is that membership in H_n can be tested efficiently: A number between 0 and $n/2$ belongs to H_n iff its Jacobi symbol is $+1$. (Hence it is also easy to test membership in $G_{\tau,n}$.) The operation in $G_{\tau,n}$ is defined by

$$(a, x) \circ (b, y) := ((a + b) \bmod 2^\tau, xy4^{(a+b)\mathrm{div}2^\tau}),$$

and the unit element of $G_{\tau,n}$ is $(0,1)$.

- The homomorphism $h_{\tau,n}$ mapping $G_{\tau,n}$ to H_n is defined by

$$h_{\tau,n}((a,x)) := \pm(4^a x^{2^\tau}),$$

i.e., either $4^a x^{2^\tau}$ or $-4^a x^{2^\tau}$, depending on which of them is smaller than $n/2$. The corresponding (unoptimized) probabilistic polynomial-time algorithm h is clear. ◇

THEOREM 4.5.

(a) *Under Assumption* F*, Construction* 4.4 *is a family of bundling homomorphisms.*

(b) *For any* $\tau \in \mathbb{N}$ *and any odd* $n \in \mathbb{N}$ *(i.e., not only for those chosen according to Construction* 4.4*),* $h_{\tau,n}$ *is a homomorphism satisfying Definition* 4.1(b)*.*

(c) *If* $n = p^r q^s$*, where* p *and* q *are primes with* $p \equiv 3$ *and* $q \equiv 7 \bmod 8$ *and* r *and* s *are odd, then for any* $a \in \mathbb{Z}_{2^\tau}$ *and* $z \in H_n$*, there exists exactly one* $x \in H_n$ *so that* $h_{\tau,n}((a,x)) = z$*. (The reason we consider numbers* n *of this more general form is that we want to use the fairly efficient zero-knowledge protocol from* [21]*; see below.)*

Proof. See [4]. The last claim is only proved for $r = s = 1$, but the same proof also works in the more general case. □

Given this family of bundling homomorphisms, we can construct a fail-stop signature scheme called the *Factoring Scheme*. We only have to fill in the three parameters that were left open in Construction 4.1:

- The message space is $\{0, \ldots, 2^\rho - 1\}$ for any $\rho \in \mathbb{N}$.
- The parameter τ for the bundling degree is computed as $\tau := \sigma + \rho$.
- A prekey n is good if it is of the form defined in Theorem 4.5(c). By Theorem 4.5(b, c), such prekeys fulfill the conditions required in Construction 4.1. The zero-knowledge proof for this predicate is as follows:

1. The center, who has chosen n and therefore knows p and q, uses the zero-knowledge protocol from [21] to prove to the signer that $n = p^r q^s$, where p and q are primes and congruent to 3 modulo 4 and r and s are odd. This proof must be executed such that the probability of cheating the signer into accepting an incorrect n is at most $2^{-\sigma}$.
2. The signer verifies on her own that $n \equiv 5 \bmod 8$ to exclude the case $p \equiv q \bmod 8$.

Given all this, it is straightforward to write down the details of the Factoring Scheme in the same manner as in Construction 4.3. The following theorem completes the security considerations.

THEOREM 4.6. *The Factoring Scheme is secure for the signer.*

Proof. According to the corollary to Theorem 4.2, it only remains to show $|T_{m'}| \leq 2^\rho$ for all m' whenever the prekey is good because then

$$|T_{m'}|/2^\tau \leq 2^{-\sigma}.$$

Note that in $G_{\tau,n}$,

$$(a,x)^{m'} = (0,1) \Rightarrow m'a \bmod 2^\tau = 0 \Rightarrow \mathrm{ord}(a) \mid m'.$$

Hence

$$T_{m'} \subseteq \{(a,x) \in G_{\tau,n} \mid h_{\tau,n}((a,x)) = 1 \wedge \mathrm{ord}(a) \mid m'\}.$$

According to Theorem 4.5(c), for each a, there is exactly one x such that $h_{\tau,n}((a,x)) = 1$. Thus

$$|T_{m'}| \le |\{a \in \mathbb{Z}_{2^\tau} | \mathrm{ord}(a) | m'\}| = \gcd(2^\tau, m').$$

By the choice of the message space, m' is between 1 and $2^\rho - 1$ (m' is the difference between two different legal messages; negative values need not be considered separately), and therefore $\gcd(2^\tau, m') < 2^\rho$. □

As for the efficiency, first note that a multiplication by 4 modulo n can be replaced by two shifts and at most two subtractions and is therefore negligible compared with a general multiplication modulo n. A group operation in $G_{\tau,n}$ is therefore essentially one modular multiplication because the exponent of 4 is 0 or 1. This yields the following:

- Signing: The term $m\,sk_2$ with $sk_2 \in G_{\tau,n}$ can be evaluated with any exponentiation algorithm. Since we just saw that a group operation in $G_{\tau,n}$ is essentially a modular multiplication, this corresponds to an exponentiation modulo n with the exponent m. In other words, it needs between ρ and $(3/2)\rho$ modular multiplications with a suitable exponentiation algorithm.
- Test: A signature $s = (a, x)$ on the message m is acceptable iff

$$pk_1\, pk_2^m = h_{\tau,n}((a,x)) \Leftrightarrow pk_1 = \pm 4^a x^{2^\tau} pk_2^{-m}.$$

Since the message is ρ bits long and $\tau = \rho + \sigma$, we see that a signature can be tested using at most $\tau + \rho = 2\rho + \sigma$ modular multiplications. With a good exponentiation algorithm, one can get close to $\rho + \sigma$ modular multiplications.

- The length of the secret key (for one message) is $2(k + \tau) = 2(k + \rho + \sigma)$ bits.
- The length of the public key is $2k$.
- The signature length (for a ρ-bit message) is $k + \tau = k + \rho + \sigma$ bits.

Thus the two schemes require almost the same amount of computation and storage for keys and signatures. The main difference is that the key generation in the first scheme is simpler. A variant of the Factoring Scheme that does not need a zero-knowledge proof, at the cost of an additional summand k in most of the lengths and numbers of multiplications, is given in [34]. Table 1 compares the two schemes for $k = \rho = |q|_2 \approx |p|_2$; remember that k is also the length of the message.

TABLE 1
Complexity of the two instantiations of the general construction.

	Discrete logarithm	Factoring
sign	2 multiplications	$\approx k$ multiplications
test	$< 2k$ multiplications	$< 2k + \sigma$ multiplications
Length of sk	$4k$	$4k + 2\sigma$
Length of pk	$2k$	$2k$
Length of a signature	$2k$	$2k + \sigma$

5. Signing many long messages. In section 4, constructions of fail-stop signature schemes for one short message were presented. We now extend these constructions so that an arbitrary number of messages of arbitrary length can be signed.

5.1. Overview. The easiest way to sign more than one message is to use a scheme from section 4 and to prepare as many keys as one intends to sign messages.

However, this is not very practical. In particular, the distribution of public keys assumes that each signer has access to a reliable broadcast channel (which may be realized by a certification hierarchy or a kind of phone book in practice). The use of such a channel should be minimized. In fact, some authors even require of ordinary digital signature schemes that after the initial key generation of fixed length, signing can go on "polynomially forever" [1, 30]. Hence one important result of the next two constructions is that they guarantee very short public keys.

The basic idea behind these constructions is tree authentication. The same ideas underlie all published provably secure ordinary digital signature schemes; however, some variations necessitate a new proof each time. Actually, there are two types of tree authentication, that of [29, 26, 27] and that of [28, 20], which we call bottom-up and top-down, respectively, corresponding to how the tree is constructed. The former leads to shorter signatures; the latter is more flexible. We sketch their fail-stop versions in sections 5.3 and 5.4, respectively.

In both of these constructions, the length of the secret key is linear in the number of messages to be signed. We prove in section 6 that this cannot be avoided if one defines "secret key" as in Definition 3.1, i.e., including all of the secret random bits that the signer will ever need. However, we show that this secret key never needs to exist completely at the same time: In section 5.4, we modify the top-down construction so that only a small amount of secret storage is needed.

In addition, we show in section 5.2 how long messages can be hashed before signing so that the length of signatures and keys is independent of the message length. In section 5.5, we present a special variant of the constructions based on discrete logarithms that shortens the secret key by approximately a factor of 2. This will be interesting in comparison with the lower bounds; see section 6. In section 5.6, we sketch more efficient constructions for the case where all the signatures by one signer have the same recipient.

5.2. Collision-resistant hash functions and message hashing. Message hashing means that a hash function is applied to each message before it is signed, which reduces messages of arbitrary length to a fixed length. At first sight, it could seem impossible to do this with fail-stop signatures. Given a correct signature s on a message m, an arbitrarily powerful forger can find another message m' that is mapped to the same string by the hash function. Hence this forger knows that s is also the correct signature on m'. Thus the signer cannot use the idea described in section 1.3 to repudiate the successful forgery (m', s). Fortunately, this is not a problem because such a forgery gives the signer a collision of the hash function, which she can present as a proof of forgery. Under the assumption that the hash function is collision-resistant, the signer cannot construct such a collision by herself.

Collision-resistant hash functions have been formally defined in [10]. Under Assumptions DL and F, efficient collision-resistant hash functions exist that need about one multiplication per message bit and where the length of the output is the length of the modulus used in the hash function [10, 8]. In practice, one may decide to use much faster hash functions whose security cannot be proved under well-known assumptions (or cannot even be defined, e.g., if they have no keys), such as RIPEMD-160 [13] (follow the references for more such functions and known attacks).

If a hash function is used in a fail-stop signature scheme, the key for the hash function, i.e., the particular instance of the hash function, must be chosen by the center (or the recipients according to section 3.3), i.e., in our constructions, it is part of the prekey. The parameters must be chosen so that the output of the hash function

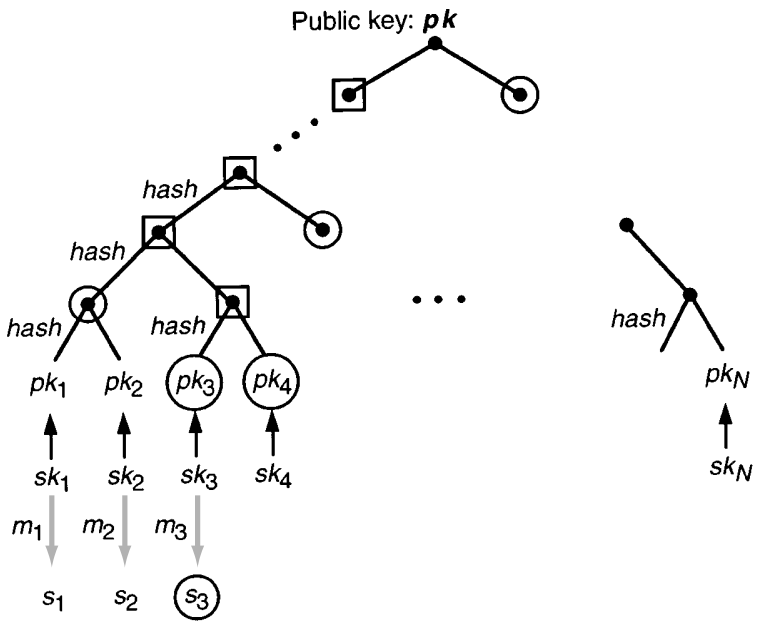

FIG. 2. *Fail-stop signature scheme with bottom-up tree authentication. Thin black arrows denote the relation between a one-time secret key and the corresponding one-time "public" key, broad grey arrows denote one-time signatures, and the tree is constructed by repeatedly hashing pairs of values. For instance, the complete signature on m_3 consists of the encircled nodes. To test it, the recipient reconstructs the nodes in squares.*

fits into the message length of the underlying fail-stop signature scheme. Then it is easy to prove that such a combination of a secure collision-resistant hash function and a secure fail-stop signature scheme is a secure fail-stop signature scheme for messages of arbitrary length.

5.3. Bottom-up tree authentication. The construction of bottom-up tree authentication is sketched in Figure 2. It is based on any collision-resistant hash function and any one-time fail-stop signature scheme with prekey (all constructions in section 4 are of this form). We distinguish signatures and keys in the underlying one-time FSS scheme and the scheme now to be constructed by calling them *one-time* and *complete*, respectively, and put one-time "public" keys in quotes because they are no longer public.

A prekey and an instance of the hash function are chosen once and for all (by the center or the recipients). The signer generates N one-time key pairs (sk_j, pk_j) based on this prekey. She hashes them pairwise in the form of a binary tree. Only the final hash value, i.e., the root of the tree, is published as the complete public key pk of the new scheme.

The complete signature on the jth message, m_j, starts with the one-time signature s_j on m_j using sk_j. Secondly, it contains the corresponding value pk_j. Moreover, to authenticate pk_j, the branch from pk_j to the root is needed; thus the complete signature also contains the other children of the nodes on this branch (see Figure 2). Hence its length is logarithmic in N.

The recipient tests the one-time signature s_j using pk_j, reconstructs the values on the path to the root, and tests if this path ends at the correct public key pk.

A proof of forgery in the new scheme is either a proof of forgery in the one-time scheme or a collision of the hash function.

A complete formal description and a proof can be found in [35]. Here we concentrate on top-down tree authentication instead.

5.4. Top-down tree authentication and a construction with a small amount of secret storage. In this section, we proceed in two steps:

1. We present a natural fail-stop version of top-down tree authentication. It can be based on any one-time FSS scheme with prekey where arbitrarily long messages can be signed and thus, in particular, on the schemes from section 4 together with message hashing. In this construction, the public key is short and only a small amount of secret storage is needed immediately after the keys are chosen. However, a long secret key accumulates as more and more messages are signed.

2. We add measures so that the amount of secret storage is also small all the time. These measures are constructed specifically for the general construction described in section 4.2, i.e., for the efficient schemes based on factoring and discrete logarithms.

As in section 5.3, we use *one-time* and *complete* to distinguish signatures and keys in the underlying scheme and in the scheme to be constructed.

Construction 5.1 (top-down tree authentication). Let a one-time FSS scheme with prekey for the message space $\{0,1\}^+$ be given (see Definition 4.2). We construct a scheme for signing N messages as follows (see Figure 3).

- Key generation:
  - A prekey *prek* for the one-time FSS scheme is chosen and the protocol (P, V) is carried out for it.
  - The signer generates one one-time key pair (sk, pk) based on *prek* and publishes pk (and N, if it is not globally fixed) as the complete public key of the new scheme.
- Signing: Signing takes place in a binary tree with N leaves. The nodes are denoted by bit strings j and labeled with one-time key pairs (sk_j, pk_j). The root is Node ε and labeled with (sk, pk). The children of Node j are Node $j0$ and Node $j1$. All one-time key pairs are randomly generated by the signer based on the same prekey *prek*. A one-time secret key at an inner node is used to sign the pair of one-time "public" keys of its two children; a one-time secret key at a leaf is used to sign a real message. A complete signature s in the new scheme is one branch of these one-time signatures. To sign the first real message $m_{0\ldots0}$, only the one-time keys on the leftmost branch and their immediate other children have to be generated. Figure 3 shows the situation after $m_{0\ldots0}$ has been signed. Generally, a complete signature on m_j consists of the one-time signature on m_j using sk_j and the sequence of pairs of one-time "public" keys with their one-time signatures on the path from m_j to the root (see Figure 3). Thus the tree is gradually constructed from left to right.
- Test: Given a complete signature s, i.e., a branch of a tree, the recipient tests all the one-time signatures in it and checks that the path ends at the correct public key pk.
- Proof of forgery: Given a successfully forged complete signature s^*, i.e., a branch that ends at pk, the signer finds a node j where it "links in" to the correct tree, i.e., the same pk_j is used in s^* and in her current tree, but the message m_j^* signed at this node in s^* has not been signed by her. (Depending on the position of Node j, m_j^* may be a real message or a pair of one-time "public" keys.) She computes a proof of forgery, $proof_j$, for this forged one-time signature s_j^* in the one-time scheme. The complete proof of forgery, *proof*, consists of $proof_j$, pk_j, m_j^*, and s_j^*.

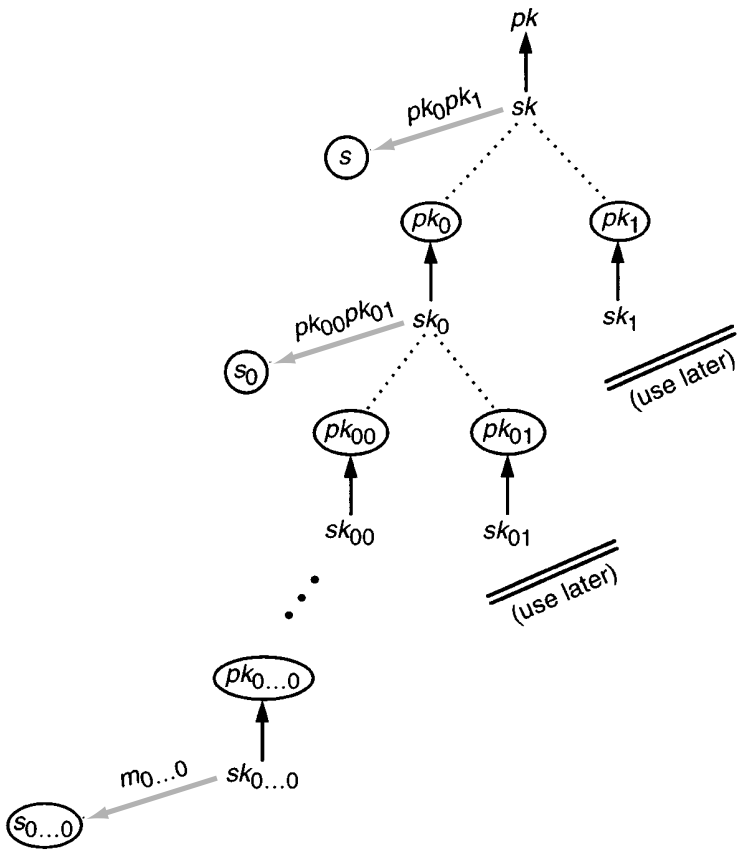

FIG. 3. *Fail-stop signature scheme with top-down tree authentication. Thin black arrows denote the relation between a one-time secret key and the corresponding one-time "public" key, broad grey arrows denote one-time signatures, and dotted lines only indicate a tree but are not related to a computation. For instance, the complete signature on* $m_{0\ldots0}$ *consists of the encircled nodes.*

• Verification of a proof of forgery: Given a value $proof = (proof_j, pk_j, m_j^*, s_j^*)$, verify that it is a valid proof of forgery in the one-time scheme, i.e., compute $verify(pk_j, m_j^*, s_j^*, proof_j)$. ◇

THEOREM 5.1. *If the underlying one-time FSS scheme is secure, Construction* 5.1 *describes a secure FSS scheme for* N *messages.*

Proof. The requirement from Definition 3.1 is obviously fulfilled. The security for the recipients, i.e., the fact that a polynomially bounded signer cannot construct signatures that she can later prove to be forgeries, follows immediately from that of the one-time scheme because any valid proof of forgery in the complete scheme is also a valid proof of forgery in the one-time scheme. (Remember that repeated use of the same prekey does not weaken the security; see the remarks after Definition 4.2.)

As to the security for the signer, first note that the probability that the signer accepts a bad prekey is the same in the one-time and the complete scheme. We show that all outcomes of the key generation based on a good prekey are in $Good_{\tilde{C}}$. For any successful forgery, the forger must select at least one node j where his complete signature links in to the correct tree, i.e., where he must forge a one-time signature. Given the forgery, the signer will find such a node (with probability 1) and can prove

the forgery at that node with the same probability as in the one-time scheme. □

A large amount of secret storage is still needed in Construction 5.1 because all the values sk_j must be stored secretly so that forgeries at any node can be proved. The basic idea used to reduce secret storage in Construction 5.2 is to store those sk_j's that are no longer used for signing in encrypted form (reliably, but not secretly) and to store only the encryption key secretly. However, information-theoretically secure encryption is needed, and a one-time pad (which is the only absolutely secure encryption scheme) is of no use because the key would be just as long as the encrypted message [42]. Hence special care must be taken that each individual sk_j remains secret, although information about the ensemble of sk_j's may become known.

We show how this can be done for the general Construction 4.1, taking into account that a forger has a priori information about the encrypted sk_j's because he may know the corresponding one-time "public" keys and signatures.

Construction 5.2 (scheme with small amount of secret storage). Let a one-time FSS scheme according to Construction 4.1 together with message hashing be given, and apply top-down tree authentication (Construction 5.1) to it with the following modifications:

- In key generation, the signer additionally chooses a value $e \in G$ randomly as an encryption key. She keeps e secret all the time.
- In signing: Whenever the signer has used up a one-time secret key $sk_j = (sk_{j,1}, sk_{j,2})$ by signing a message m_j (which may be a real message or a pair of one-time "public" keys), she proceeds as follows:
  - She encrypts $sk_{j,2}$ as $c_j := sk_{j,2} + e$.
  - She stores m_j, the one-time signature s_j, and the ciphertext c_j reliably but not necessarily secretly.
- For a proof of forgery: If the signer needs a one-time secret key sk_j to prove a forgery, she reconstructs it as follows: She decrypts $sk_{j,2} = c_j - e$ and recomputes $sk_{j,1} = s_j - m\, sk_{j,2}$, where $m := hash(m_j)$. ◇

THEOREM 5.2. *If the underlying one-time FSS scheme is secure, Construction* 5.2 *describes a secure FSS scheme for N messages. At any time, only the encryption key e and the one-time secret keys that have been generated but not yet used for signing (marked "use later" in Figure* 3*), i.e., at most one per level of the tree, must be stored secretly.*

Proof. In addition to Theorem 5.1, we only have to show that the extra information stored nonsecretly, i.e., the ciphertexts c_j, does not help a forger to produce unprovable forgeries.

In every successfully forged complete signature, the signer still finds at least one successfully forged one-time signature s_j^* on a message m_j^*. If m_j^* and a message m_j that the signer has signed at this node are a collision of the hash function, this collision is already a proof of forgery. Otherwise, Theorem 4.1(c) guarantees that the forgery can be proved if s_j^* is different from the correct one-time signature that the signer would have produced for m_j^* at Node j. Thus it remains to be shown that the additional information does not help a forger to find exactly this signature. This signature depends only on sk_j, i.e., not on the values sk_l at other nodes. We consider the worst case, where the signer has already signed a message m_j at Node j. Let $m := hash(m_j)$. In the one-time scheme, the set of possible values sk_j from the point of view of a forger was $SK_{\bar{C},j} = \{(s_j - m\, sk'_{j,2}, sk'_{j,2}) | h(sk'_{j,2}) = pk_{j,2}\}$ with uniform distribution (see the proof of Theorem 4.2). Hence it suffices to show that all of these values are still possible if the forger has seen c_j and all the other ciphertexts c_l.

Let such a value $sk'_{j,2}$ be given. It corresponds to exactly one encryption key $e' := c_j - sk'_{j,2}$. This implies that the other one-time secret keys must be given by

$$sk'_{l,2} := c_l - e' = sk'_{j,2} + c_l - c_j.$$

The only question is whether these are possible secret keys, i.e., whether $h(sk'_{l,2}) = pk_{l,2}$. On one hand,

$$h(sk'_{l,2}) = h(sk'_{j,2}) \cdot h(c_l)/h(c_j) = pk_{j,2} \cdot h(c_l)/h(c_j),$$

and on the other hand,

$$h(c_l) = h(sk_{l,2}) \cdot h(e) = pk_{l,2} \cdot h(e) \quad \text{and} \quad h(c_j) = pk_{j,2} \cdot h(e).$$

Hence $h(sk'_{l,2}) = pk_{l,2}$. □

If this construction is applied to a usual complete tree, the amount of information that must be stored secretly is logarithmic in N (see Table 2 in section 6.5 for an example).

If we use a list-shaped tree, i.e., the left child of each node is a real message, at any time only two sk_j's have been generated but not yet used for signing. Thus the amount of secret storage is constant (as a function of N). However, later signatures are very long. Thus list-shaped trees should only be used with a single recipient; see section 5.6.

One can also use trees of other shapes, e.g., ternary trees or trees where one decides dynamically whether the message m_j signed at a node is a real message or a pair of one-time "public" keys and specifies this by one additional bit in m_j.

5.5. Improvement for the discrete logarithm constructions. In this section, we present a fail-stop signature scheme for signing N messages where the total length of the secret key is $(2N + 2)k$. The smallest upper bound in the previous sections was $4Nk$, achieved by pure repetition of the one-time Discrete Logarithm Scheme and also in bottom-up tree authentication based on it.

The scheme is a variant of pure repetition of the Discrete Logarithm Scheme where half of each one-time secret key is reused in the next signature.

Construction 5.3.

- Key generation:
  - As in Construction 4.3, a prekey (p, q, a, b) is chosen and verified, where p and q are primes with $q|(p-1)$ and a and b are generators of the group H_q, the unique subgroup of $\mathbb{Z}_p^*$ of order q.
  - The signer chooses a secret key

$$sk := ((x_1, y_1), (x_2, y_2), \ldots, (x_{N+1}, y_{N+1}))$$

consisting of numbers between 0 and $q - 1$. The corresponding public key is

$$pk := (pk_1, \ldots, pk_{N+1}) := (a^{x_1} b^{y_1}, \ldots, a^{x_{N+1}} b^{y_{N+1}}).$$

- The message space is restricted to $\mathbb{Z}_q^*$.
- Signing: The correct signature on the jth message, m_j, is the triple

$$s := (j, x_j + m_j x_{j+1}, y_j + m_j y_{j+1}).$$

- Test: A triple (j, x, y) is an acceptable signature on the jth message, m_j, iff
$$a^x b^y = pk_j pk_{j+1}^{m_j}.$$
- Proofs of forgery are constructed and verified as in Construction 4.3. ◇

THEOREM 5.3. *Under Assumption* DL*, Construction* 5.3 *describes a secure FSS scheme.*

Proof. It is clear that correct signatures pass the test because individual signatures are identical to those in Construction 4.3. Similarly, the security for the recipients is fulfilled because proofs of forgery are identical to those in Construction 4.3. Furthermore, every successful forgery f that is not the correct signature, i.e., $f = (m_j^*, (j, x, y))$ with $(j, x, y) \neq sign(sk, j, m_j^*)$, can be proved as before. It remains to show that the reuse of halves of the one-time secret keys does not make it too easy for a forger to guess correct signatures.

It suffices to show this for the worst case, where the signer has already issued N signatures on messages $m_1, \ldots, m_N$, possibly chosen by the forger, and the forger therefore has maximum information. We first determine the size of the set $SK_{\tilde{C}}$ of possible secret keys given these signatures and the public key. $SK_{\tilde{C}}$ is the set of solutions to the equations defining the public key and the signatures. With $e := \log_a(b)$, all of these equations can be written as linear equations over GF(q). They are described by the following $(3N+1) \times (2N+2)$ matrix A. The columns correspond to the elements of sk in the order as above, the first $2N$ rows to the signatures, and the last $N+1$ rows to the components of the public key.

$$A = \begin{pmatrix}
1 & 0 & m_1 & & & & & & & \\
 & 1 & 0 & m_1 & & & & & & \\
 & & 1 & 0 & m_2 & & & & & \\
 & & & 1 & 0 & m_2 & & & & \\
 & & & & & \ddots & & & & \\
 & & \mathbf{0} & & & & & & & \\
 & & & & & & 1 & 0 & m_N & \\
 & & & & & & & 1 & 0 & m_N \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
1 & e & & & & & & & & \\
 & & 1 & e & & & & & & \\
 & & & & & \ddots & & & & \\
 & & \mathbf{0} & & & & & & & \\
 & & & & & & & & 1 & e
\end{pmatrix}.$$

We now show that the rank of A is $2N+1$. For this, consider the following submatrix of A:

$$\begin{pmatrix}
1 & 0 & m_j & 0 \\
0 & 1 & 0 & m_j \\
1 & e & 0 & 0 \\
0 & 0 & 1 & e
\end{pmatrix}.$$

One can easily see that this submatrix has rank 3 and that the third row can be removed. By repeating this step for $j := 1, \ldots, N$, we delete all rows $(0 \ldots 0\, 1\, e\, 0 \ldots 0)$ except for the last one. The resulting matrix clearly has rank $2N + 1$.

Hence $|SK_{\bar{C}}| = q$. We now show that each of these q possible secret keys yields a different correct signature on the new message m_j^*. For this it suffices to show that any additional signature determines the secret key uniquely. The additional signature introduces two new equations corresponding to the rows

$$\begin{array}{ccccccc} 0 \ldots 0 & 1 & 0 & m_j^* & 0 & 0 \ldots 0 \\ 0 \ldots 0 & 0 & 1 & 0 & m_j^* & 0 \ldots 0 \end{array}.$$

One can easily see that the rank of the new matrix is $2N+2$. (Remember that $m_j \neq 0$ for all j.) Hence an additional, forged signature is correct only for one out of the q possible secret keys. This completes the proof. □

Construction 5.3 can be combined with bottom-up tree authentication; this yields public keys as short as in section 5.3 and secret keys as short as in Construction 5.3.

5.6. The case of a single recipient. In some important applications, all signatures of a signer have the same recipient, while any third party is able to settle a dispute (otherwise, one would not need a signature scheme). In this case, significant efficiency improvements are possible because the recipient can store information from previously received signatures. Now the most efficient construction is top-down tree authentication with a list-shaped tree so that only one new node of the tree must be produced, sent, and tested each time. Thus signing and testing are reduced to the following procedures:

- Signing: Assume that the current one-time secret key is sk_j. The signer generates a new one-time key pair (sk_{j+1}, pk_{j+1}); this can be done off-line before the new message m_j is known. The complete signature on m_j consists of the new one-time "public" key pk_{j+1} and a one-time signature with sk_j on the pair (m_j, pk_{j+1}).
- Test: The recipient tests the new one-time signature with his current one-time "public" key pk_j and then sets pk_{j+1} as his current one-time "public" key.

The old one-time secret and "public" keys can be stored as in Construction 5.2. If a third party has to settle a dispute, it must test the signature with respect to the original public key pk. This can be done with overhead logarithmic in the length of the list: With logarithmic search, one first finds an index $j' \leq j$ where the signer and the recipient agree on $pk_{j'}$ but not on the pair $(m_{j'}, pk_{j'+1})$ signed with respect to it. Then it suffices to solve the dispute regarding this one-time signature.

6. Lower bounds on the efficiency of fail-stop signatures. In this section, we try to answer the following two questions:

1. How close to the optimum are the existing constructions?

2. Is there a certain price in efficiency to pay for information-theoretic security instead of computational security?

We do this by giving lower bounds on the length (more precisely, the entropy) of keys and signatures. In practice, these parameters correspond to the communication and storage complexity of both key generation and the transmission of signed messages. Similar lower bounds have been investigated for other cryptographic schemes with information-theoretic security requirements, e.g., encryption schemes [42], au-

thentication codes (starting with [16]; see, e.g., [40]), and unconditionally secure signature schemes [23].[3]

Section 6.1 sketches the information-theoretic background, section 6.2 introduces the random variables, section 6.3 investigates the length of the secret key, section 6.4 investigates the length of signatures and public keys, and section 6.5 answers the questions posed above, i.e., it compares the lower bounds to the known upper bounds and to ordinary digital signature schemes.

6.1. Information-theoretic background. Since a random variable with entropy σ cannot be coded with less than σ bits on average, it is sufficient to prove lower bounds on the entropy of random variables in order to have lower bounds on the length of the values of these random variables.

For the formal theorems and proofs, we assume that the reader is familiar with elementary information theory (see [41] and [15, sections 2.2 and 2.3]). We briefly repeat the most important notions in the notation of [15]. Assume that a common probability space is given. Capital letters denote random variables and small letters their corresponding values, and $\mathrm{P}(X = x)$ is abbreviated as $\mathrm{P}(x)$, etc. The joint random variable of X and Y is written as X, Y. The entropy of a random variable X is

$$\mathrm{H}(X) := -\sum_x \mathrm{P}(x) \log \mathrm{P}(x).$$

$\mathrm{H}(X \mid Y)$ denotes the conditional entropy of X if Y is known, and $\mathrm{I}(X;Y)$ denotes the mutual information between X and Y. They are defined as follows:

$$\mathrm{H}(X \mid Y) : = -\sum_{x,y} \mathrm{P}(x,y) \log \mathrm{P}(x \mid y),$$

$$\mathrm{I}(X;Y) := \mathrm{H}(X) - \mathrm{H}(X \mid Y).$$

Furthermore, the conditional mutual information is defined as

$$\mathrm{I}(X;Y \mid Z) := \mathrm{H}(X \mid Z) - \mathrm{H}(X \mid Y,Z).$$

The following important rules will often be used:

$$\mathrm{H}(X) \geq 0, \qquad \mathrm{H}(X \mid Y) \geq 0, \tag{6.1}$$

$$\mathrm{H}(Y,Z \mid X) = \mathrm{H}(Y \mid X) + \mathrm{H}(Z \mid Y,X), \tag{6.2}$$

$$\mathrm{I}(X;Y) = \mathrm{I}(Y;X). \tag{6.3}$$

Additionally, we often need Jensen's inequality for the special case of the logarithm [14]: If $p_i \geq 0$ and $x_i > 0$ for all i and the sum of the p_i's is 1, then

$$\log\left(\sum_i p_i x_i\right) \geq \sum_i p_i \log(x_i).$$

[3] However, the lower bounds in the following cannot be derived entirely from information-theoretic security requirements; this makes some new proof techniques necessary.

6.2. The random variables. The common probability space for all of the random variables arising in a fail-stop signature scheme is given by the random bits used in key generation; here we made use of the fact that we could define all of the other algorithms as deterministic by regarding all random bits as part of the secret key.

For the lower bounds, we only need the case where all parties execute G honestly, and we always consider a fixed triple of parameters $par = (k, \sigma, N)$. Then the probabilities of sk and pk are uniquely determined. Hence we can define the following random variables:

- SK and PK are the random variables of the secret and public key, respectively.
- If a message sequence $\underline{m} = (m_1, \ldots, m_j)$ with $j \leq N$ has been fixed, random variables S_i for $i = 1, \ldots, j$ are defined as the correct signature on m_i in this context, i.e.,

$$S_i := sign(SK, i, (m_1, \ldots, m_i)).$$

Since the signature is a deterministic function of the secret key and the message sequence, we have

$$\mathrm{H}(S_i \mid SK) = 0.$$

- Similarly, for the given message sequence $\underline{m} = (m_1, \ldots, m_j)$, the random variable of the history up to the ith signature (for $i \leq j$) is defined as

$$Hist_i := ((m_1, \ldots m_i), (S_1, \ldots, S_i)).$$

6.3. Lower bound on the number of secret random bits needed. As mentioned in section 5, the largest difference between fail-stop signatures and ordinary digital signatures is the length of the secret key. We have shown in Theorem 5.2 that the signer need not store a lot of secret information at the same time; however, the overall number of secret random bits chosen was still linear in the number of messages to be signed. We now show that this cannot be avoided.

If all fail-stop signature schemes followed the construction idea in section 1.3, this would be quite easy to see:

1. Even an arbitrarily powerful forger must not be able to guess the signer's correct signature.
2. Hence the entropy of the correct signature is large.
3. Since this holds for each additional signature, even if some signatures are already known, the entropies of the signatures can be added, and therefore the overall entropy of the signer's secrets is large.

Our proof follows this outline; however, only a weaker average version of statement 1 can be derived from the definitions. If we required that applying *prove* to the signer's correct signature never yielded a valid proof of forgery, a formal proof would be easy and would yield

$$\mathrm{H}(SK \mid PK) \geq (N+1)\sigma.$$

However, e.g., in fail-stop signature schemes with message hashing, an arbitrarily powerful forger sometimes *does* know correct signatures on new messages (those with the same hash value as the original message), and the signer can prove them to be forgeries.

Furthermore, note that the desired lower bound cannot possibly be proved from the signer's security alone. As a counterexample, suppose that the signer were allowed

to disavow all signatures in an ordinary digital signature scheme; then she would be unconditionally secure without using many random bits (but the recipient would not be secure at all). This is a problem because Definition 3.2, like all computational cryptographic definitions, is asymptotic, i.e., security is only guaranteed for "k sufficiently large." Thus in a certain sense, we can only derive lower bounds for $k \geq k_0$ for an unknown k_0. This may seem unsatisfactory: No one would have doubted that we need arbitrarily long keys if we made k sufficiently large.

However, the real purpose of lower bounds is to say "whenever we have certain requirements on the security, we have to pay the following price in terms of efficiency." In this section, this means, more precisely, "if the signer wants the probability of unprovable forgery to be at most $2^{-\sigma}$, and the recipients want some security, too, then at least the following number of random bits is needed (as a function of σ and the security for the recipients)."

To quantify the security for the recipients, it suffices for our purpose to consider the probability that the signer can prove that her own signatures are "forgeries" simply by applying the algorithm *prove*. In practice, one will require this probability to be at most, say, 2^{-100}, or, more generally, $2^{-\sigma^*}$ for some σ^*. We will prove the lower bounds as a function of this parameter σ^* (in addition to σ). Now we can proceed formally.

DEFINITION 6.1. *Let a fail-stop signature scheme and parameters σ and N be given. For every message sequence $\underline{m} = (m_1, \ldots, m_{N+1})$, we define a polynomial-time algorithm $\tilde{A}_{\underline{m}}$ that describes how a dishonest signer could try to disavow her own correct signatures. (This algorithm should be rather useless!)*

1. *Execute G correctly (with the center) to obtain sk and pk.*

2. *Compute the signatures $s_1, \ldots, s_N$ on $m_1, \ldots, m_N$ and the histories $hist_1, \ldots, hist_N$ correctly. Then because m_{N+1} is one message too much, compute its signature s_{N+1} as if m_N had not been signed, i.e., $s_{N+1} := sign(sk, N, (m_1, \ldots, m_{N-1}, m_{N+1}))$.*

3. *If $provable(sk, pk, hist_{i-1}, (m_i, s_i))$ for an $i \in \{1, \ldots, N+1\}$, then output the proof $prove(sk, m_i, s_i, hist_{i-1})$.*

We say that k is large enough to provide security level σ^* for the recipients against $\tilde{A}_{\underline{m}}$ *if the success probability of $\tilde{A}_{\underline{m}}$ is at most $2^{-\sigma^*}$.*[4] *This probability is over the random bits of $\tilde{A}_{\underline{m}}$ and the random bits of the center in step 1.*

THEOREM 6.1. *Let a fail-stop signature scheme with actual parameters σ and N and a security level σ^* be given. Let $\sigma' := \min(\sigma, \sigma^*)$. Then for all k large enough to provide security level σ^* for the recipients against the algorithm $\tilde{A}_{\underline{m}}$ for at least one sequence $\underline{m}$ of $N+1$ pairwise-distinct messages,*[5]

$$\mathrm{H}(SK \mid PK) \geq (N+1)(\sigma' - 1).$$

[4]The formal definition of the recipient's security immediately implies the existence of k_0 such that $\tilde{A}_{\underline{m}}$ has this property for all $k \geq k_0$. We have now bypassed the problem that we do not know how large k_0 is because we simply know that it must be large enough in a practical application.

Note that it would be impossible to require k to be large enough to provide the same level of security against all polynomial-time algorithms instead of only $\tilde{A}_{\underline{m}}$ because for a fixed k, we have a finite problem, and all finite problems can be solved perfectly by some constant-time algorithm. However, our $\tilde{A}_{\underline{m}}$ is not only asymptotically polynomial-time but definitely feasible at the given k because it is of the same complexity as *sign* and *prove*. Requirements of the same type are well known in practice, e.g., when one requires k to be large enough to make the success probability of all known factoring algorithms small.

[5]Note that this is a very weak condition on k. The contrary is that the signer can sign an arbitrary message sequence $\underline{m}$ and disavow it with significant probability using $\tilde{A}_{\underline{m}}$.

Since $\underline{m}$ is fixed throughout the proof, we can omit it in the notation, i.e., all random variables S_i and $Hist_i$ are implicitly assumed to belong to this message sequence. We also omit $\underline{m}$ in the histories, i.e., we abbreviate $Hist_i := (S_1, \ldots, S_i)$ for $i = 1, \ldots, N+1$. The first lemma formalizes statement 1 above, i.e., that it is hard to guess correct signatures.

LEMMA 6.1. *With the notation of Theorem* 6.1*, for each* $i \leq N+1$ *and each pair* $(pk, hist_{i-1})$*, fix an optimal guess* $s^*(pk, hist_{i-1})$ *at the correct signature on* m_i *if* pk *and* $hist_{i-1}$ *are known, i.e., for all values* s*,*

$$\mathrm{P}(S_i = s^*(pk, hist_{i-1}) \mid pk, hist_{i-1}) \geq \mathrm{P}(S_i = s \mid pk, hist_{i-1}).$$

Then this guess is still not very good on average:

$$\mathrm{P}(S_i = s^*(PK, Hist_{i-1})) \leq 2^{-\sigma'+1}.$$

Proof. Let i and $(pk, hist_{i-1})$ be given. We have

$$\begin{aligned}
&\mathrm{P}(S_i = s^*(PK, Hist_{i-1})) \\
&\quad = \mathrm{P}(S_i = s^*(PK, Hist_{i-1}) \wedge provable(SK, PK, Hist_{i-1}, (m_i, S_i))) \\
&\qquad + \mathrm{P}(S_i = s^*(PK, Hist_{i-1}) \\
&\qquad\quad \wedge \neg provable(SK, PK, Hist_{i-1}, (m_i, s^*(PK, Hist_{i-1})))) \\
&\quad \leq \mathrm{P}(provable(SK, PK, Hist_{i-1}, (m_i, S_i))) \\
&\qquad + \mathrm{P}(\neg provable(SK, PK, Hist_{i-1}, (m_i, s^*(PK, Hist_{i-1})))).
\end{aligned}$$

The first term is bounded by the success probability of $\tilde{A}_{\underline{m}}$ and therefore by $2^{-\sigma^*}$. The second term can be bounded using the security for the signer (Definition 3.4):

$$\begin{aligned}
&\mathrm{P}(\neg provable(SK, PK, Hist_{i-1}, (m_i, s^*(PK, Hist_{i-1})))) \\
&= \sum_{pk, hist_{i-1}} \mathrm{P}(pk, hist_{i-1}) \mathrm{P}(\neg provable(SK, PK, Hist_{i-1}, (m_i, s^*(PK, Hist_{i-1}))) \\
&\qquad\qquad \mid PK = pk, Hist_{i-1} = hist_{i-1}) \\
&\leq \sum_{pk, hist_{i-1}} \mathrm{P}(pk, hist_{i-1}) 2^{-\sigma} \\
&= 2^{-\sigma}.
\end{aligned}$$

Together, these two bounds yield the lemma. □

Lemma 6.2 uses Lemma 6.1 to give a lower bound corresponding to statement 2 of the informal proof sketch.

LEMMA 6.2. *With the notation of Theorem* 6.1*, for each* $i \leq N+1$*,*

$$\mathrm{H}(S_i \mid PK, Hist_{i-1}) \geq \sigma' - 1.$$

Proof. We can write the conditional entropy as

$$\mathrm{H}(S_i \mid PK, Hist_{i-1}) = -\sum_{s_i,pk,hist_{i-1}} \mathrm{P}(s_i, pk, hist_{i-1}) \log(\mathrm{P}(s_i \mid pk, hist_{i-1}))$$

$$\geq -\log\left(\sum_{s_i,pk,hist_{i-1}} \mathrm{P}(s_i, pk, hist_{i-1})\mathrm{P}(s_i \mid pk, hist_{i-1})\right)$$

using Jensen's inequality. (Note that one *cannot* prove that the individual values $\log(\mathrm{P}(s_i \mid pk, hist_{i-1}))$ are at least σ.) Now it suffices to show that the sum in the logarithm is at most $2^{-\sigma'+1}$. In fact, with Lemma 6.1,

$$\sum_{s_i,pk,hist_{i-1}} \mathrm{P}(s_i, pk, hist_{i-1})\mathrm{P}(s_i \mid pk, hist_{i-1})$$

$$\leq \sum_{s_i,pk,hist_{i-1}} \mathrm{P}(s_i, pk, hist_{i-1})\mathrm{P}(S_i = s^*(pk, hist_{i-1}) \mid pk, hist_{i-1})$$

$$= \sum_{pk,hist_{i-1}} \left(\mathrm{P}(S_i = s^*(pk, hist_{i-1}) \mid pk, hist_{i-1}) \sum_{s_i} \mathrm{P}(s_i, pk, hist_{i-1})\right)$$

$$= \mathrm{P}(S_i = s^*(PK, Hist_{i-1}))$$

$$\leq 2^{-\sigma'+1}. \qquad \square$$

Proof of Theorem 6.1. Theorem 6.1 can be proved from Lemma 6.2 using rule (2) from section 6.1. Recall the abbreviation $Hist_i = (S_1, \ldots, S_i)$ for $i \leq N+1$. First, we show by induction over i that the entropy of all of these signatures together is large, i.e., for all $i \leq N+1$,

$$\mathrm{H}(Hist_i \mid PK) \geq i(\sigma' - 1). \tag{4}$$

For $i = 1$, (4) is Lemma 6.2. If (4) has already been proved for $i-1$, it holds for i because

$$\begin{aligned}\mathrm{H}(Hist_i \mid PK) &= \mathrm{H}(Hist_{i-1} \mid PK) + \mathrm{H}(S_i \mid PK, Hist_{i-1}) \\ &\geq (i-1)(\sigma'-1) + (\sigma'-1) \\ &= i(\sigma'-1).\end{aligned}$$

We now use the fact that signing is deterministic, i.e., SK uniquely determines $Hist_{N+1}$. This implies $\mathrm{H}(Hist_{N+1} \mid PK, SK) = 0$, and therefore

$$\begin{aligned}\mathrm{H}(SK \mid PK) &= \mathrm{H}(SK, Hist_{N+1} \mid PK) - \mathrm{H}(Hist_{N+1} \mid PK, SK) \\ &= \mathrm{H}(SK, Hist_{N+1} \mid PK) \\ &\geq \mathrm{H}(Hist_{N+1} \mid PK) \\ &\geq (N+1)(\sigma'-1). \qquad \square\end{aligned}$$

6.4. Length of signatures and public keys. Signatures and public keys are not much longer in current fail-stop signature schemes than in ordinary digital signature schemes. Hence the lower bounds are also very small. The basic idea why fail-stop signatures might be longer at all is as follows:

1. First, there must be at least 2^σ acceptable signatures; otherwise, the correct signature could be guessed too easily.

2. Second, it must be hard for a forger to guess acceptable signatures at all. Thus the density of the set of acceptable signatures within the signature space should be small, e.g., at most $2^{-\sigma^*}$ for some σ^*.

Hence we expect the size of the signature space to be at least $2^{\sigma+\sigma^*}$. We will prove a slightly lower bound but more generally on the entropy of each signature.

We deal with the asymptotic character of the security against forgery in the same way as in section 6.3, i.e., we fix one simple algorithm that a forger might use to guess acceptable signatures.

DEFINITION 6.2. *Let a fail-stop signature scheme and parameters σ and N be given. For every message sequence $\underline{m} = (m_1, \ldots, m_i)$ with $i \leq N$, we define a (very efficient and stupid) polynomial-time forging algorithm $\tilde{F}_{\underline{m}}$. On input pk, do the following:*

1. *Choose a new key pair (sk^*, pk^*) by carrying out both roles of G correctly.*

2. *Compute a signature $s_i := sign(sk^*, i, \underline{m})$ with the new key and output (m_i, s_i) as a proposed forgery for the given key pk.*

We say that k is large enough to provide security level σ^* against forgery by $\tilde{F}_{\underline{m}}$ *if the probability that $\tilde{F}_{\underline{m}}$ outputs an acceptable signature on m_i is at most $2^{-\sigma^*}$.*[6]

THEOREM 6.2. *Let a fail-stop signature scheme with actual parameters σ and N and a security level σ^* be given. Let $\sigma' := \min(\sigma, \sigma^*)$. Then for all k large enough to provide security level σ^* both against forgery by $\tilde{F}_{\underline{m}}$ and against $\tilde{A}_{\underline{m}}$ for at least one sequence $\underline{m}$ of $i \leq N$ pairwise-distinct messages,*

$$\mathrm{H}(S_i) \geq \sigma' + \sigma^* - 1$$

and

$$\mathrm{H}(PK) \geq \sigma^*.$$

The following lemma formalizes step 2 of the informal proof sketch. The fact that the number of acceptable signatures, given the public key, is much smaller than the complete signature space is generalized as follows: The public key contains a lot of information about the correct signature.

LEMMA 6.3. *With the notation of Theorem* 6.2*,*

$$\mathrm{I}(S_i; PK) \geq \sigma^*.$$

[6] The security against forgery, Theorem 3.1, implies the existence of k_0 such that all $k \geq k_0$ have this property.

Proof. By the definitions and Jensen's inequality,

$$\begin{aligned}
\mathrm{I}(S_i; PK) &= - \sum_{s_i, pk : \mathrm{P}(s_i, pk) \neq 0} \mathrm{P}(s_i, pk) \log(\mathrm{P}(s_i)\mathrm{P}(pk)/\mathrm{P}(s_i, pk)) \\
&\geq - \log \left(\sum_{s_i, pk : \mathrm{P}(s_i, pk) \neq 0} \mathrm{P}(pk, s_i)\mathrm{P}(s_i)\mathrm{P}(pk)/\mathrm{P}(s_i, pk) \right) \\
&= - \log \left(\sum_{s_i, pk : \mathrm{P}(s_i, pk) \neq 0} \mathrm{P}(s_i)\mathrm{P}(pk) \right).
\end{aligned}$$

It suffices to show that the argument of the logarithm is bounded above by $2^{-\sigma^*}$. First, the fact that all correct signatures pass the test, i.e.,

$$\mathrm{P}(s_i, pk) \neq 0 \Rightarrow test(pk, m_i, s_i) = ok,$$

yields

$$\begin{aligned}
\sum_{s_i, pk : \mathrm{P}(s_i, pk) \neq 0} \mathrm{P}(s_i)\mathrm{P}(pk) &= \sum_{s_i, pk} \mathrm{P}(s_i)\mathrm{P}(pk) 1_{\mathrm{P}(s_i, pk) \neq 0} \\
&\leq \sum_{s_i, pk} \mathrm{P}(pk)\mathrm{P}(s_i) 1_{test(pk, m_i, s_i) = ok}.
\end{aligned}$$

Now we use the security against $\tilde{F}_{\underline{m}}$, which can be written formally as

$$\begin{aligned}
\mathrm{P}(test(pk, m_i, s_i) = ok :: (acc, pk, sk, \varepsilon) \leftarrow G; (acc^*, pk^*, sk^*, \varepsilon) \leftarrow G; \\
s_i := sign(sk^*, i, \underline{m})) \leq 2^{-\sigma^*}.
\end{aligned}$$

This means that if a public key pk is chosen according to the distribution of PK and, independently of this, a signature is chosen according to the distribution of S_i, this signature is acceptable with respect to pk with probability at most $2^{-\sigma^*}$. Hence

$$\sum_{s_i, pk} \mathrm{P}(pk)\mathrm{P}(s_i) 1_{test(pk, m_i, s_i) = ok} \leq 2^{-\sigma^*}.$$

This is exactly what remained to be shown. □

Proof of Theorem 6.2. Lemma 6.3 implies $\mathrm{H}(S_i) - \mathrm{H}(S_i \mid PK) \geq \sigma^*$, and a special case of Lemma 6.2 implies $\mathrm{H}(S_i \mid PK) \geq \sigma' - 1$. (Here we exploit the security against $\tilde{A}_{\underline{m}}$.) As a consequence, $\mathrm{H}(S_i) \geq \mathrm{H}(S_i \mid PK) + \sigma^* \geq \sigma' + \sigma^* - 1$. Furthermore, $\mathrm{H}(PK) \geq \mathrm{I}(S_i; PK)$. □

For the case with a prekey (see section 4.2), we obtain slightly stronger results by using a forging algorithm $\tilde{F}^*_{\underline{m}}$ that generates its new key pair (sk^*, pk^*) based on the same prekey *prek*. In this case, the same proof yields

$$\mathrm{H}(S_i \mid Prek) \geq \sigma' + \sigma^* - 1 \quad \text{and} \quad \mathrm{H}(PK \mid Prek) \geq \sigma^*.$$

If, as usual in such schemes, pk is a function of sk and *prek*, we obtain one more result about the secret key.

TABLE 2
Comparison of constructions and lower bounds for fail-stop signatures.

Length of	Construction 5.1	Construction 5.2	Construction 5.3	Bottom-up	Single recipient	Lower bound
sk	$8Nk$	$8Nk$	$(2N+2)k$	$4Nk$	$4Nk$	$(N+2)(\sigma'-1)$
secret storage	$8Nk$	$4k\log(N)$	$(2N+2)k$	$4Nk$	$10k$	?
pk	$2k$	$2k$	$(N+1)k$	$2\sigma^*$	$2k$	σ^*
signature	$6k\log(N)$	$6k\log(N)$	$2k+\log(N)$	$6k+2\sigma^*\log(N)$	$4k$	$\sigma'+\sigma^*-1$

THEOREM 6.3. *In a fail-stop signature scheme with prekey and where the public key is a function of the secret key and the prekey, with the notation of Theorems* 6.1 *and* 6.2*,*

$$\mathrm{H}(SK \mid Prek) \geq (N+2)(\sigma'-1).$$

Proof. By rule (2), we get

$$\begin{aligned}\mathrm{H}(SK \mid Prek) &= \mathrm{H}(SK, PK \mid Prek) - \mathrm{H}(PK \mid SK, Prek)\\ &= \mathrm{H}(PK \mid Prek) + \mathrm{H}(SK \mid PK, Prek) - 0\\ &= \mathrm{H}(PK \mid Prek) + \mathrm{H}(SK \mid PK).\end{aligned}$$

In the last line, we used the fact that the prekey is part of the public key. Hence

$$\mathrm{H}(SK \mid Prek) \geq \sigma^* + (N+1)(\sigma'-1) \geq (N+2)(\sigma'-1). \qquad \square$$

6.5. Comparison. This section answers the two questions raised at the beginning of this section by evaluating how well our constructions meet the lower bounds and by comparing the lower bounds with similar bounds on ordinary digital signature schemes.

Table 2 approximately shows the length of the secret key (including all of the random bits needed later), the maximal amount of secret storage needed, the length of the public key and the length of a signature for Constructions 5.1, 5.2, and 5.3, bottom-up tree authentication, and the case of a single recipient. The table also shows the corresponding lower bounds; remember that all schemes use a prekey. All constructions are assumed to be based on the Discrete Logarithm Scheme from Construction 4.3 with parameters as in Table 1. Remember that $2k$ bits are needed to represent an element of the group G, k bits are needed for an element of H, and the total number of one-time secret keys in the tree in Figure 3 is $2N-1$. The bottom-up construction is shown for the case where the recipients trust a fast hash function with output length $2\sigma^*$.

It should be mentioned that [22] also presents a scheme for signing N messages where the signature length is only $2k$, and the key lengths and secret storage are the same as in Construction 5.3. However, the complexity of signing then grows linearly in N, and $\log(N)$ will be dominated by k anyway.

The difference between k in the upper bounds and σ and σ^*, and thus $\sigma' = \min(\sigma, \sigma^*)$, in the lower bounds is as follows: Usually, σ and σ^* would be chosen of equal size, say 100, whereas k must be at least 500 to give sufficient computational

security. If one believes that variants of the discrete logarithm problem exist that cannot be solved in subexponential time, e.g., on nonsupersingular elliptic curves (cf. [24]), one can choose k very close to σ^*, and then the lower bounds are met by the upper bounds except for small factors.

By disregarding the difference between k, σ, and σ^*, the current upper bound on the length of the secret key is $(2N+2)\sigma$ in contrast to the lower bound $(N+2)(\sigma-1)$. For the signature length, both bounds are 2σ, and for the public key, the upper bound is 2σ and the lower bound σ. However, having both public keys and signatures of length independent of N was only achieved if the signatures of each signer have a single recipient (or very few recipients). If every new signature may have a new recipient, there is a tradeoff: If the public key is of constant length, the signatures grow logarithmically in N because of the tree authentication. It is an interesting open question whether this can be avoided.

If we compare fail-stop signatures with ordinary digital signatures, the most obvious difference in terms of complexity is that the number of secret random bits needed for fail-stop signatures is linear in the number of messages to be signed. The lower bound on the length of fail-stop signatures is $\sigma'+\sigma^*-1$; for ordinary digital signatures, σ^* is a lower bound. Hence the two types of schemes differ by a factor of about 2. For public keys, the same lower bound σ^* holds for ordinary digital signature schemes, and in fact, even in the constructions, public keys of fail-stop signature schemes are not longer than in ordinary digital signature schemes.

7. Conclusion and outlook. We have defined fail-stop signature schemes formally and shown that their security is in fact stronger than that of ordinary digital signature schemes. In particular, one can use fail-stop signatures either as ordinary digital signatures or as signatures with dual security, where the signers instead of the recipients are secure in an information-theoretic sense. Based on the definition, we proved general lower bounds on the length of the keys and signatures in these schemes.

We presented a general construction of fail-stop signature schemes and showed how to instantiate this construction under either of the well-known assumptions that factoring large integers or computing discrete logarithms is hard. These constructions come quite close to the lower bounds; see Table 2.

There are some recent additional results about fail-stop signature schemes. In [11], it was shown that fail-stop signature schemes exist if statistically hiding bit-commitment schemes of a certain type exist. This implies that they exist if one-way permutations exist, which is a potentially weaker assumption than the existence of claw-free pairs of permutations (see section 1.4). Furthermore, a practical construction from arbitrary collision-resistant hash functions was given, and the existence of certain types of fail-stop signature schemes and bit-commitment schemes was shown to be equivalent. In [33], the open question from section 6.5, whether a fail-stop signature scheme exists where the length of both public keys and signatures is independent of the number N of messages to be signed, was answered positively under a strong computational assumption (with a construction based on the one-way accumulators from [3]).

In conjunction with a theorem from [23] which shows that the length of unconditionally secure signatures (see section 1.5) is linear in the number of participants who can test them (whereas the length of fail-stop signatures and ordinary digital signatures does not depend on this number), our constructions show that fail-stop signatures provide the most viable way of protecting signers unconditionally.

Acknowledgments. It is a pleasure to thank many people for their contributions, primarily, Eugène van Heijst, who was a coauthor of preliminary versions of parts of this paper and would have been a coauthor again if he were still working in this field, and Michael Waidner, who allowed us to use some material from joint publications here. We also thank Ivan Damgård and Andreas Pfitzmann for several good ideas and Joachim Biskup, Gerrit Bleumer, David Chaum, and the anonymous referees for interesting comments.

REFERENCES

[1] M. Bellare and S. Micali, *How to sign given any trapdoor permutation*, J. Assoc. Comput. Mach., 39 (1992), pp. 214–233.

[2] J. Benaloh, *Verifiable secret-ballot elections*, Ph.D. thesis, Yale University, New Haven, CT, 1987.

[3] J. Benaloh and M. de Mare, *One-way accumulators: A decentralized alternative to digital signatures*, in Proc. 1993 Eurocrypt, Lecture Notes in Comput. Sci. 765, Springer-Verlag, Berlin, 1994, pp. 274–285.

[4] G. Bleumer, B. Pfitzmann, and M. Waidner, *A remark on a signature scheme where forgery can be proved*, in Proc. 1990 Eurocrypt, Lecture Notes in Comput. Sci. 473, Springer-Verlag, Berlin, 1991, pp. 441–445.

[5] J. Boyar, D. Chaum, I. Damgård, and T. Pedersen, *Convertible undeniable signatures*, in Proc. 1990 Crypto, Lecture Notes in Comput. Sci. 537, Springer-Verlag, Berlin, 1991, pp. 189–205.

[6] J. Bos, D. Chaum, and G. Purdy, *A voting scheme*, unpublished manuscript, presented at the rump session of 1988 Crypto, Santa Barbara, CA, 1988.

[7] D. Chaum and H. van Antwerpen, *Undeniable signatures*, in Proc. 1989 Crypto, Lecture Notes in Comput Sci. 435, Springer-Verlag, Berlin, 1990, pp. 212–216.

[8] D. Chaum, E. van Heijst, and B. Pfitzmann, *Cryptographically strong undeniable signatures, unconditionally secure for the signer*, in Proc. 1991 Crypto, Lecture Notes in Comput. Sci. 576, Springer-Verlag, Berlin, 1992, pp. 470–484.

[9] D. Chaum and S. Roijakkers, *Unconditionally secure digital signatures*, in Proc. 1990 Crypto, Lecture Notes in Comput. Sci. 537, Springer-Verlag, Berlin, 1991, pp. 206–214.

[10] I. B. Damgård, *Collision free hash functions and public key signature schemes*, in Proc. 1987 Eurocrypt, Lecture Notes in Comput. Sci. 304, Springer-Verlag, Berlin, 1988, pp. 203–216.

[11] I. B. Damgård, T. P. Pedersen, and B. Pfitzmann, *On the existence of statistically hiding bit commitment schemes and fail-stop signatures*, in Proc. 1993 Crypto, Lecture Notes in Comput. Sci. 773, Springer-Verlag, Berlin, 1994, pp. 250–265.

[12] W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Trans. Inform. Theory, 22 (1976), pp. 644–654.

[13] H. Dobbertin, A. Bosselaers, and B. Preneel, *RIPEMD*-160: *A strengthened version of RIPEMD*, in Proc. 3rd Fast Software Encryption Workshop, Lecture Notes in Comput. Sci. 1039, Springer-Verlag, Berlin, 1996, pp. 71–82.

[14] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. II, 2nd ed., John Wiley, New York, 1971.

[15] R. Gallager, *Information Theory and Reliable Communication*, John Wiley, New York, 1968.

[16] E. N. Gilbert, F. J. MacWilliams, and N. J. A. Sloane, *Codes which detect deception*, Bell System Technical J., 53 (1974), pp. 405–424.

[17] O. Goldreich, *Two remarks concerning the Goldwasser–Micali–Rivest signature scheme*, in Proc. 1986 Crypto, Lecture Notes in Comput. Sci. 263, Springer-Verlag, Berlin, 1987, pp. 104–110.

[18] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems*, J. Assoc. Comput. Mach., 38 (1991), pp. 691–729.

[19] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–207.

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

[21] J. van de Graaf and R. Peralta, *A simple and secure way to show the validity of your public key*, in Proc. 1987 Crypto, Lecture Notes in Comput. Sci. 293, Springer-Verlag, Berlin, 1988, pp. 128–134.

[22] E. van Heyst and T. P. Pedersen, *How to make efficient fail-stop signatures*, in Proc. 1992 Eurocrypt, Lecture Notes in Comput. Sci. 658, Springer-Verlag, Berlin, 1993, pp. 366–377.

[23] E. van Heijst, T. P. Pedersen, and B. Pfitzmann, *New constructions of fail-stop signatures and lower bounds*, in Proc. 1992 Crypto, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 15–30.

[24] N. Koblitz, *Elliptic curve implementation of zero-knowledge blobs*, J. Cryptology, 4 (1991), pp. 207–213.

[25] L. Lamport, *Constructing digital signatures from a one-way function*, Report CSL-98, SRI International, Menlo Park, CA, 1979.

[26] R. C. Merkle, *Protocols for public key cryptosystems*, in Proc. 1980 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, CA, 1980, pp. 122–134.

[27] R. C. Merkle, *Protocols for public key cryptosystems*, in Secure Communications and Asymmetric Cryptosystems, G. J. Simmons, ed., AAAS Selected Symposium 69, Westview Press, Boulder, CO, 1982, pp. 73–104.

[28] R. C. Merkle, *A digital signature based on a conventional encryption function*, in Proc. 1987 Crypto, Lecture Notes in Comput. Sci. 293, Springer-Verlag, Berlin, 1988, pp. 369–378.

[29] R. C. Merkle, *A certified digital signature* (*That antique paper from* 1979), in Proc. 1989 Crypto, Lecture Notes in Comput. Sci. 435, Springer-Verlag, Berlin, 1990, pp. 218–238.

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

[31] B. Pfitzmann, *Für den Unterzeichner unbedingt sichere digitale Signaturen und ihre Anwendung*, Diploma thesis, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Karlsruhe, Germany, 1989 (in German; summarized in English in [35]).

[32] B. Pfitzmann, *Fail-stop signatures: Principles and applications*, in Proc. 8th World Conference on Computer Security, Audit, and Control (1991 Compsec), Elsevier, Oxford, UK, 1991, pp. 125–134.

[33] B. Pfitzmann, *Fail-stop signatures without trees*, Hildesheimer Informatik-Berichte 16/94 (ISSN 0941-3014), Institut für Informatik, Universität Hildesheim, Hildesheim, Germany, 1994.

[34] B. Pfitzmann, *Digital Signature Schemes: General Framework and Fail-Stop Signatures*, Lecture Notes in Comput. Sci. 1100, Springer-Verlag, Berlin, 1996.

[35] B. Pfitzmann and M. Waidner, *Formal aspects of fail-stop signatures*, Technical Report 22/90, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1990.

[36] B. Pfitzmann and M. Waidner, *Fail-stop signatures and their application*, in Proc. 9th Worldwide Congress on Computer and Communications Security and Protection (1991 Securicom), Paris, 1991, pp. 145–160.

[37] B. Pfitzmann and M. Waidner, *Unconditional Byzantine agreement for any number of faulty processors*, in Proc. 9th Annual Symposium on Theoretical Aspects of Computer Science (1992 STACS), Lecture Notes in Comput. Sci. 577, Springer-Verlag, Berlin, 1992, pp. 339–350.

[38] R. L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. Assoc. Comput. Mach., 21 (1978), pp. 120–126; reprinted in Comm. Assoc. Comput. Mach., 26 (1983), pp. 96–99.

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

[40] R. Safavi-Naini and L. Tombak, *Optimal authentication systems*, in Proc. 1993 Eurocrypt, Lecture Notes in Comput. Sci. 765, Springer-Verlag, Berlin, 1994, pp. 12–27.

[41] C. E. Shannon, *Communication in the presence of noise*, Proc. Inst. Radio Engineers, 37 (1949), pp. 10–21.

[42] C. E. Shannon, *Communication theory of secrecy systems*, Bell System Technical J., 28 (1949), pp. 656–715.

[43] M. Waidner and B. Pfitzmann, *The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability*, in Proc. 1989 Eurocrypt, Lecture Notes in Comput. Sci. 434, Springer-Verlag, Berlin, 1990, p. 690.