Securicom '91, Paris, 19.-22. March 1991, 145-160.

Signatures dont des falsifications sont prouvables, et leur application

Fail-stop Signatures and their Application

Birgit Pfitzmann, Michael Waidner*

Abstract. The unforgeability of conventional digital signatures is necessarily based on complexity theoretic assumptions, i.e. even the most secure schemes can be broken by an adversary with unexpected computing abilities. Thus we introduce fail-stop signatures: They are as unforgeable as the best conventional signatures, but if a signature is forged nevertheless, the supposed signer can prove the forgery unconditionally (i.e. without assumptions), with arbitrarily high probability.

We construct actual fail-stop signature schemes, called hiding schemes, from arbitrary claw-free pairs of permutations. As a special case, we obtain a rather practical system where forging is as hard as factoring.

We also present applications to digital payment systems, and sketch those to reliable broadcast.

* Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Postfach 6980, D-W7500 Karlsruhe 1, Fed. Rep. of Germany; Phone: ++49-721-608-4024, Fax: ++49-721-370455, E-mail (CSnet): Waidner@Ira.UKa.De

1 Introduction

Motivation: Digital signatures are an important cryptographic primitive [DiHe_76]. Their distinguishing feature is that a recipient of a signature can convince everybody else of its validity (in contrast to authentication codes [GiMS_74], where only the recipient is sure that a message was sent by a particular person). Hence, digital signatures are particularly necessary in all applications where legal certainty is required, i.e. where signed messages may have to be shown to a court. An example is digital payment systems.

The security of digital signatures has recently received some attention. This started in particular because the best-known system, RSA [RSA_78] in its pure form is known to be vulnerable by active attacks. This means one can compute new signatures from previously obtained ones, without breaking the secret key (see, e.g., [Denn_84]). Therefore, redundancy or hash functions are used; but even less is known about the security of those than about that of RSA itself [DaPr_85, EkHV_88, Gira_88, JoCh_86]. The same problems occur with the system from [Elga_85], cf. [GoMR_88]. Some other systems have been broken completely (see, e.g., [Odly_84, EAKM_86, BrDe_86]).

In [GoMR_88], the optimal security for (conventional) digital signatures was therefore defined, and systems with this security were constructed. By a **conventional signature scheme** we mean one according to their definition. (This definition comprises RSA with a hash function, only its security could not be proved.) In particular, each signer Alice publishes a public key PK, and a string S counts as Alice's signature under a message M iff it passes a test $test(PK, M, S)$. This implies that forging signatures is within NP.

Additionally, all known conventional signature schemes are based on generally trusted, but unproved complexity theoretic assumptions. The original proposal from [GoMR_88], called GMR, is based on the existence of claw-free pairs of trapdoor permutations. This assumption could first be weakened to any trapdoor functions [BeMi_88], and then to any one-way functions [NaYu_89]. To use signatures in practice, one must pick out a specific function which is hopefully one-way. Normally, one assumes the intractability of integer factoring or of some discrete log. (The most efficient scheme is still that based on factoring from [GoMR_88], with an overhead comparable to that of the better known, but not provably secure, RSA [RSA_78].)[1]

Fail-stop signatures improve the security in a different way: They unconditionally enable signers to prove forgeries (with arbitrarily high probability). **Unconditionally** means that no assumption must be made, not even that an adversary can only execute polynomial time algorithms. Advantages are:

- The scheme can be stopped, or security parameters increased, as soon as forgeries occur.
- The risk of forged signatures can be divided arbitrarily between signers and recipients. In particular, if signatures become invalid if a proof of forgery has been shown, the signers are unconditionally secure.

---

[1] Although it is not needed in the following, note the advantage of a provably secure signature scheme: It has been proved that if factoring is hard, no polynomial time algorithm has a non-negligible chance of forging any GMR-signature in any context; whereas nothing similar has been proved for RSA.

We also require that, on cryptographical assumptions, forgeries cannot occur at all, and signers cannot falsely claim that a forgery occurred. Thus, the security definition of fail-stop signatures is strictly stronger than that of a conventional signature scheme.

REMARKS: There have been proposals for conventional signatures, saying that signers notice if forgeries occur, or if the secrecy of their keys is lost, and are then allowed to stop the scheme. However, that would mean total insecurity for the recipients of signatures, since a signer could just claim that a forgery had occurred to shuffle off the responsibility for a signature.

Fail-stop signatures have additional practical advantages in public applications like payment systems: First, it is doubtful if people can be forced to take part in systems where they must rely on cryptographical assumptions, with security parameters which they usually cannot choose. Using fail-stop signatures and conventional signatures together (see Ch. 3), one can make clients of such systems unconditionally secure. Secondly, with conventional signatures, even if the cryptographical assumption is true, clients might irrefutably claim that their signatures have been forged, which could lead to considerable public uncertainty.

Related work: Parallel to this work, "unconditionally secure signatures" have been developped, i.e. signatures where both signer and recipient are unconditionally secure [ChRo_90]. This is a beautiful idea, but its differences to conventional signatures are quite large: Each participant has a different test function. The test also depends on via how many recipients the signature has been received (i.e. whether directly from the signer, or from the first recipient, etc.). Moreover, active attacks on recipients may be harmful. Therefore, in contrast to fail-stop signatures, unconditional signatures need special protocols where everybody knows their number of predecessors, and where active attacks on recipients are restricted. The main scheme of [ChRo_90] corresponds to the case with just one risk-bearer here (see Ch. 2). It is not yet known how large the signatures must be in the general case. (The tricks from [ChRo_90, Ch. 5] do not help if the signer attacks.) In many cases, fail-stop signatures will be more efficient than unconditional signatures. In particular, the key exchange for unconditional signatures is rather a complex protocol.

A precise security definition and a proof of our basic fail-stop signature schemes can be found in [Pfit_89, PfWa_90], our protocols for reliable broadcast (and a first, but inefficient fail-stop signature scheme) in [WaPf_89, PfWa1_91]. A special aspect of the key generation has been discussed in [Bleu_90, BlPW_90].

2 Construction of Fail-Stop Signatures

2.1 Basic idea

The basic idea for the construction of fail-stop signatures is that, given a public key PK, there are many possible secret keys SK which could belong to it; and that for each message M, these different secret keys yield many different possible signatures (even after an adaptive chosen message attack, i.e. even if the adversary has received several signatures under messages of his choice). However, a signer has just one secret key SK^*, and can compute just one signature S^* efficiently. Presenting two different signatures for the same message M and the same public key PK is considered as a proof of forgery.[2] One easily sees that a scheme with these properties is a secure fail-stop signature scheme:

1. Even if a forger has *unlimited* computing power, the most he can do is compute all possible signatures for a message M. However, he can only guess which one is S^*. With high probability, he guesses wrong and chooses another signature S. Thus as soon as the real signer is confronted with the forgery S (to hold her responsible for it), she can present both S and S^*, which counts as a proof of forgery.
2. On the other hand, it will be shown that, even for the signer, it is *computationally* infeasible to compute a proof of forgery if no previous forgery has occurred, on a cryptographical assumption such as the hardness of factoring. Thus, as long as this assumption is true, the signature scheme works just like a conventional signature scheme, without any proofs of forgery appearing. (This implies that forgeries don't appear either, since those, according to Point 1., would lead to proofs of forgery.)

That there are "many" possible signatures means, e.g., 2^σ, where the computational overhead of the signature scheme is only polynomial in σ.

2.2 An actual fail-stop signature scheme: The hiding scheme

In the following, we informally present a particular fail-stop signature scheme, called *hiding scheme*. It is based on LAMPORT's *one-time signatures* [DiHe_76, Merk_88] and a new special kind of one-way functions called *bundling functions*. Also, several efficiency improvements to one-time signatures by MERKLE were adapted so that they enable both proofs of forgery and cryptographical security. The construction will be shown step by step.

2.2.1 Building blocks: Bundling functions and one-time signatures

Conventional one-time signatures: Conventional one-time signatures were the very first proposal for a digital signature scheme. They enable someone to sign an a priori fixed number of bits, say m. First the signer chooses a one-way function f, normally by choosing a key K for a function family (such as DES).

In the following, we will write such keys as upper indices. Thus f^K denotes the function f with the key K.

As the secret key SK, the signer now chooses a list of $2 \bullet m$ random elements from the domain of f^K, say

$$SK = ((r_{1,0}, r_{1,1}),\ldots, (r_{m,0}, r_{m,1})).$$

She computes the images $pk_{i,b} := f^K(r_{i,b})$. The public key consists of K and the list of these images:

$$PK = (K, (pk_{1,0}, pk_{1,1}),\ldots, (pk_{m,0}, pk_{m,1})).$$

To sign that the i-th bit has a particular value $b \in \{0, 1\}$, the signer publishes $r_{i,b}$. (Thus the secret key is just the list of all potential bit-signatures.) The recipient of a triple (i, b, s), where s is claimed to be the signature (i.e. s should be $r_{i,b}$), tests if $f^K(s) = pk_{i,b}$.

---

2 In actual schemes, parts of these two signatures will suffice as proof of forgery. For example, in a system where forging is as hard as factoring, the proof of forgery may directly be the previously unknown factors of a certain number.

Definition and use of bundling functions: We first want to construct one-time signatures with the possibility of proving forgeries. For this, we define and construct a special kind of one-way functions: Bundling functions are functions where each image has at least 2^σ preimages, but which are collision-free in a cryptographical sense, i.e. it is cryptographically hard to find two preimages with the same image.[3]

If we use a bundling function as f in one-time signatures, the public key PK gives an adversary no information about which of the at least 2^σ preimages the signer has originally chosen as $r_{i,b}$ and used in computing the image $pk_{i,b}$. By forging a signature, the adversary computes a preimage for at least one $pk_{i,b}$ where the real signer has not yet published $r_{i,b}$ (because she has either not signed the i-th bit at all so far, or wanted it to have the other value $1-b$).[4] Hence, as a signature for this bit, the forger chooses a preimage $r_{i,b}^* \neq r_{i,b}$, with probability at least $1\text{-}2^{-\sigma}$. Now as soon as the forgery is shown to the signer (to hold her responsible for it), she obtains an f-collision, namely $(r_{i,b}^*, r_{i,b})$ with the same image $pk_{i,b}$ under f^K. This counts as a proof of forgery.

Conversely, the collision-freeness immediately implies that, on the cryptographical assumption, no proof of forgery can ever be computed. However, this presupposes that in this scheme, in contrast to conventional one-time signatures, K has not been chosen by the signer, see Section 2.2.3.

Within the signature scheme, we will call the key K of the bundling function the **prekey**, because it is chosen before the actual secret and public key. The choice of K depends on the parameter σ and a second security parameter k for the cryptographical security (i.e. a larger k implies more secure collision-freeness, but also longer computations). A sensible choice for σ would be a value between 20 and 100. The functions must be efficiently computable.

Construction of bundling functions: We start by constructing bundling functions from arbitrary claw-free pairs of permutations [GoMR_88]. Using particular claw-free pairs of permutations, we then obtain rather efficient bundling functions whose collision-freeness is based on the hardness of factoring.

First we sketch the definition of **claw-free pairs of permutations**: This is a family of algorithms; one of them is a key generation algorithm G. If G has been used to generate a key K, then $f_0{}^K$ and $f_1{}^K$ are permutations on the same domain, say D^K. One can also choose random elements from D^K efficiently. Claw-freeness means that it is infeasible (for someone who did not construct K) to find two elements x, y with $f_0{}^K(x) = f_1{}^K(y)$ (although such pairs exist).

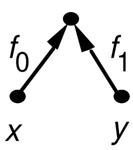

Figure 1: Claw of a claw-free pair of permutations (that what is infeasible to find, on a cryptographical assumption).

We obtain a **bundling function** *bund* with the same key K by iterating $f_0{}^K$ and $f_1{}^K$: A preimage r will consist of two parts (*string*, x); the first part is a bit string of length σ, say

---

3 Such functions were already informally defined and used in [ChFN_90], but no construction was given.

4 This is true since the contrary would be that the forger just uses a real signature received from the signer.

$string = (b_1,\ldots,b_\sigma)$, the second an element from D^K. We compute the image $pk = bund^K(r) = bund^K((string, x))$ by using the bits from *string*, one after the other, to indicate which of $f_0{}^K$ and $f_1{}^K$ to apply next:

$$bund^K(r) := f_{b_1}{}^K(f_{b_2}{}^K(\ldots(f_{b_\sigma}{}^K(x))\ldots)).$$

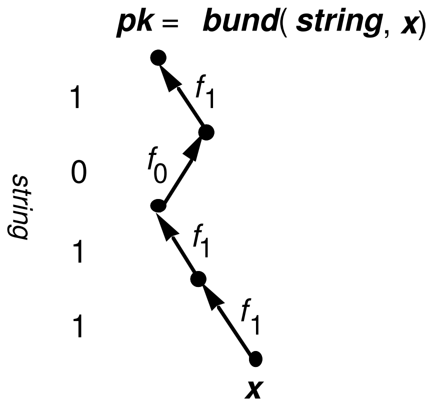

Figure 2: Example of the bundling function: $\sigma = 4$, *string* = 1011.

(The construction resembles the basic functions in [GoMR_88] and the hash functions in [Damg_88], but each time, the parameters are used differently.)

Security (Sketch): On the one hand, it is easy to see that each image *pk* has 2^σ preimages: There are 2^σ strings of length σ. Since $f_0{}^K$ and $f_1{}^K$ are permutations, for each such value *string*, there is an x_{string} with $bund^K((string, x_{string})) = pk$. In Fig. 2, this means that, starting with *pk* and working backwards, one can construct a complete binary tree of depth σ.

On the other hand, we must show that such collisions cannot be found efficiently. We prove this by contradiction: Assume that one could find collisions. We show that in this case, one could also find claws of the claw-free pair efficiently (cf. Fig. 3): A *bund*-collision means that we have two paths which meet, at the latest, at *pk*. At the place where the strings differ for the first time (starting from *pk*) there is a claw of $f_0{}^K$ and $f_1{}^K$:

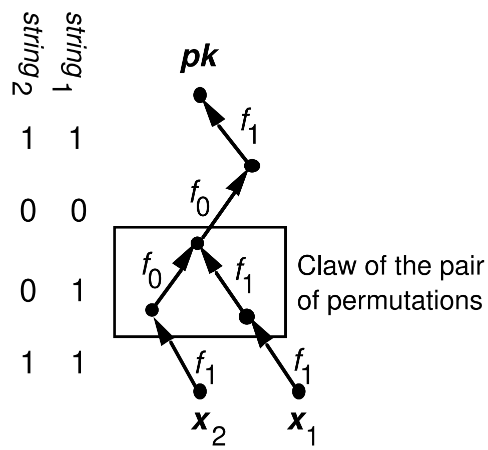

Figure 3: Example of how a collision of the bundling function would lead to a claw of f_0 and f_1.

Special case where forging is as hard as factoring: The most efficient claw-free pairs of permutations are, so far, those from [GoMR_88]: Keys are numbers $K = p \cdot q$ where

p and q are primes and $p \equiv 3$, $q \equiv 7 \bmod 8$. If one represents the residue classes modulo K symmetrically, i.e. by $\{-\frac{K-1}{2}, \ldots, +\frac{K-1}{2}\}$, and defines the absolute value for them as usual, then we have

$$f_0{}^K(x) := |x^2|$$
$$f_1{}^K(x) := |4x^2|$$

on a subset of these residue classes (the positive ones whose JACOBI symbol is +1). It has been proved that each claw could be used to factor K very easily.

If one applies $f_0{}^K$ and $f_1{}^K$ several times in succession, as within a bundling function, one need not take the absolute value of intermediate results (since the next operation is squaring again, which is equal for $\pm y$). One would not actually represent the residue classes symmetrically, so as to be able to use a normal multiplication subroutine. Also note that $4x^2 = (2x)^2$; hence apart from one squaring, there is only one doubling, no multiplication. Thus the application of a bundling function mainly consists of σ squarings. This is more efficient than signing one RSA-block. The person who has chosen K can, as with RSA [QuCo_82], compute modulo p and q separately and combine the results using the Chinese remainder algorithm.

2.2.2 Efficiency improvements

The main disadvantage of pure one-time signatures is the length of the public key PK, since it must be distributed reliably outside the network (i.e. at least the signer Alice, the recipient and a court must agree upon what Alice's valid public key is). With pure one-time signatures, the length of PK is proportional to the number of bits to be signed. This disadvantage can by avoided by applying **tree authentication**, in extension of ideas from [Merk_80, Merk_82]:

The values $bund^K(r_{i,b})$ are used as leaves of a binary tree (i.e. those values which would have been the public key in pure one-time signatures). The value of each inner node is computed as the hash value of its children, using a hash-function h. Only the value of the root is published as the public key PK (see already Fig. 4). As the secret key SK, the values $r_{i,b}$ still suffice; but to save time, one will store the whole tree. Later, to authenticate a leaf value, all its forefathers and the other child of each one must be presented (i.e., they are part of the signature). We use a cryptographically collision-free hash function [Damg_88], and define that a collision of the hash-function counts as proof of forgery, too. One can see that each forgery either leads to a proof of forgery in the original one-time signatures, or to a collision of the hash-function.

In [Damg_88], cryptographically collision-free hash functions are constructed, which are similar to the bundling functions and need about one squaring per bit that is hashed. Specially for the tree authentication, where the length of the values to be hashed is known a priori, hash functions can be constructed which are more efficient by a factor of about two [Pfit_89]; this is not shown in Figure 4. One can also hash all the values $bund^K(r_{i,b})$ belonging to the bits of the same message at once (instead of combining them in pairs first), since they are always needed together.

If the messages to be signed are long, they can also be **hashed** before being signed. Again, we use cryptographically collision-free hash functions and count collisions as proof of forgery; and again, one can see that each forgery either leads to a proof of forgery in the original one-time signatures, or to a collision of the hash-function.

Figure 4 shows how the keys are formed now, and Figure 5 how a message is signed. L denotes the (fixed) length of a message after hashing. The prekey K has been omitted for simplicity.

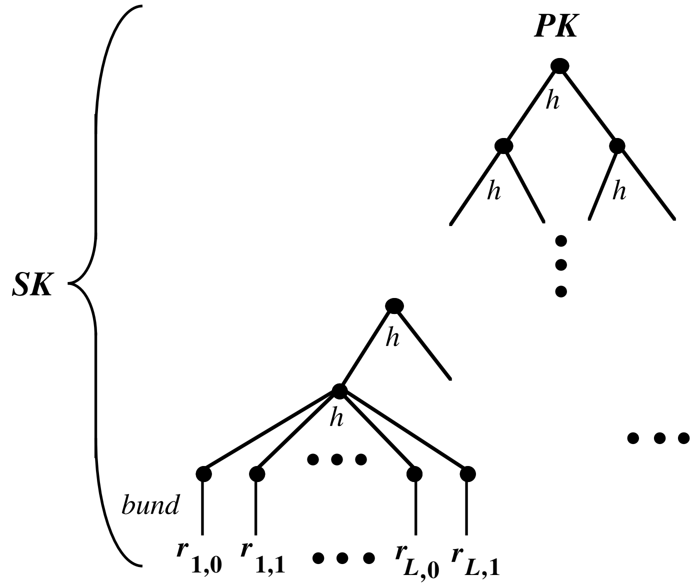

Figure 4: Keys in the hiding scheme with tree authentication and hashing

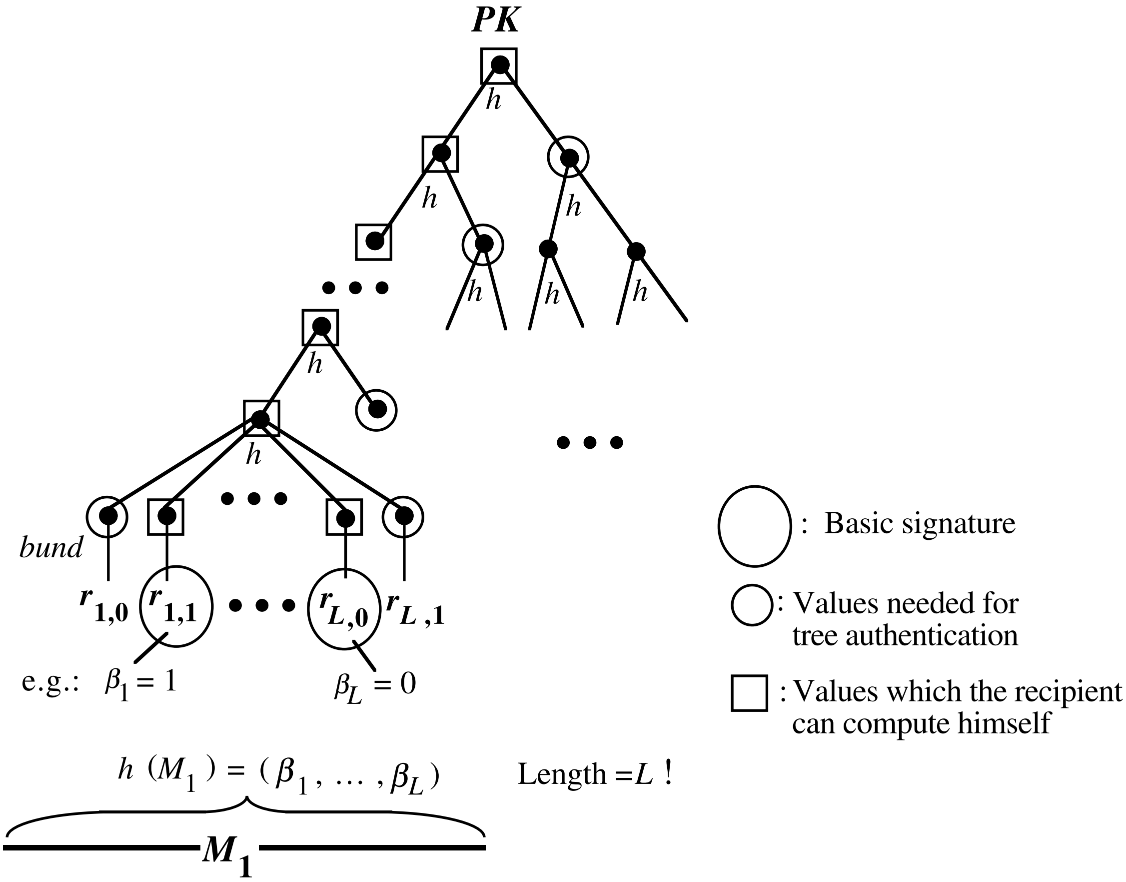

Figure 5: Signing the first message M_1 in the hiding scheme

Further efficiency improvements are: Only bits with the value 1 need to be signed if a code detecting all changes of 1 into 0 is used [Merk1_82]. The a priori limitation on the number of signatures can be avoided by using some of these signatures to authenticate new public keys, similar to [Merk_88].

2.2.3 Prekey exchange

To make collisions convincing proofs of forgery, the signer must be unable to construct them for herself efficiently. Collision-freeness, like most cryptographical properties, however, holds only against parties who cannot watch the generation of the prekey K for the bundling functions and hash functions (For instance, they must not see the primes p and q from which $K = p \bullet q$ is constructed.) Thus, instead of the signer, the party who bears the risk of a proof of forgery must choose K.

One risk-bearer: In applications like payment systems, this risk is usually borne by just one person, the intended recipient of the signature, to whom money is paid or promised (see Sect. 3.1). In this case, the intended recipient can choose K alone. (It is then not excluded that the recipient can compute a proof of forgery for the received signatures. However, if he did so, he would only harm himself, since the received signatures become invalid; e.g. he does not obtain the promised money.)

The signer's ability to prove forgeries may depend on the fact that K is a correct key for the bundling function. Thus, the intended recipient must convince the signer that K is correct, without, however, enabling the signer to construct collisions.

In the **general case**, this can be achieved by a usual "cut-and-choose" technique [Rabi_78]: The recipient generates $2 \bullet \sigma$ prekeys. The signer may choose σ of them for inspection. For those, the signer must demonstrate the correctness. The other σ prekeys are used in each signature. A collision for at least one key counts as proof of forgery. If at least one of the σ prekeys used is correct, the signer can prove forgeries with the same probability as before. The probability that all these prekeys are wrong, whereas all those selected for inspection are correct, is exponentially small. (Alternatively, a zero-knowledge proof can be used [GoMR_89, GoMW_86]. Inside it, however, the correctness of keys must be ensured, too.)

In [Bleu_90, BlPW_90], it has been proved that for our most **efficient implementation**, i.e. the special case where forging is as hard as factoring from Sect. 2.2.1, the signer may accept any odd number as K. (Thus the fact that K should consist of exactly two primes from specific residue classes modulo 8 only serves to ensure the security of the recipient.) Hence in this case, no interactive protocol at all is needed in the key exchange.

Several risk-bearers: In applications like Byzantine agreement protocols (Sect. 3.2), where several people share this risk, there are two possibilities: Either they must try to choose either K together, using a particular multiparty computation protocol [ChDG_88, PfWa_90]; however, these protocols are not very practical yet. Or each one must choose his own prekey K_i. In that case, a complete signature consists of one signature with respect to each prekey, and a proof of forgery of one collision for each prekey. Naturally, a recipient can use the same K_i for all signers.

A practically interesting case with two prekeys per signature is that where basically the recipient bears the risk of a proof of forgery alone, but the courts want to give advice about the security parameter k. Then the courts must be sure that each proof of forgery presented to them has really been produced by breaking the cryptographical assumption (e.g., by factoring a large number, not just because a recipient published the factors of his prekey for fun). For this, a public authority might choose one prekey, which should be used in all signatures. Each recipient, however, continues to use his own prekey, too, so that nobody must totally trust this public authority.

2.2.4 Resulting efficiency

We only consider our most efficient implementation (the one based on the hardness of factoring), and start by looking at the case where for each signature, there is just one risk-bearer.

As already mentioned, the most critical point is the length of the prekeys and public keys, since those must, e.g., be printed in public lists like telephone books. These are just as short as usual: K is one number of the form $p \bullet q$ which should be hard to factor, thus of a length of at least $L = 800$ bit, and PK is just one number modulo K.

Specially for 1-bit messages, the length of the signatures, and the computational expenditure of signing and testing, is almost the same as in GMR, the most efficient provably secure conventional signature scheme. However, the expenditure grows proportionally with the length of the message, until the length is reached to which the hash function reduces long messages. This length is L again. (Especially in payment systems, however, the loss in efficiency can be kept very small, see Sect. 3.1.3.)

A secret key is a bit shorter than all the signatures to be produced with it together. This is no additional problem, however, since received and produced signatures must be stored anyway (the latter, because they might be needed for proofs of forgery); and the secret key, too, must only be stored (and used sequentially in small pieces).

However, we see no theoretical reason why fail-stop signature schemes should necessarily be based on one-time signatures. It should be worthwhile looking for a scheme based on some sort of trapdoor, where signatures would have the same length, and signing and testing similar run-time, as in conventional signature schemes.

When there are more risk-bearers per signature, the overhead of the system rises considerably; for cases, where all participants share the risk of each signature, like in Byzantine agreement, the importance of fail-stop signatures is of a rather theoretical nature.

3 Applications

3.1 Payment systems

Important applications for fail-stop signatures are payment systems, which can now be made unconditionally secure for the clients. (The following also holds for all other applications where a large organization has many clients, and where signatures are only exchanged between the organization and a client, and not directly between two clients.) The organization operating the payment system will always be called a bank.

3.1.1 Use of fail-stop signatures in general

We presuppose a simple digital payment system which simulates a non-digital one with deposits into an account, withdrawals, and remittances from one account into another. It will be implemented so that the clients are unconditionally secure.

Hence we only use fail-stop signatures for the clients' signatures on their orders to the bank; the bank uses a provably secure conventional signature scheme, such as GMR, for its signatures. To ensure that no client must trust the signature of another client, each remittance is confirmed to the recipient by a signature from the bank.

Thus the bank is the only risk-bearer for the clients' fail-stop signatures (see Sect. 2.2.3); it can therefore choose the prekey K for the bundling and hash functions alone. After the bank has published K, each client can choose a secret key SK (i.e. preimages for the bundling functions in the one-time signatures) and publish the corresponding public key PK. From now on, everybody, in particular a court, can test what are valid signatures and what is a valid proof of forgery. The clients are now unconditionally secure against forgeries, and the bank is cryptographically secure that the clients cannot produce proofs of forgeries, since it has chosen K itself. (As mentioned in Sect. 2.2.3, the bank may be able to compute proofs of forgery for K, but it would only harm itself, since the orders for remittances and withdrawals would become invalid, whereas the confirmations of remittances and deposits would remain valid.)

With respect to the signatures received form the bank, i.e. the confirmations of remittances and deposits, the clients are also unconditionally secure (every signature they receive from the bank is valid forever), and the bank is cryptographically secure that no forgery occurs.

3.1.2 Advantages

It is sensible to distribute the risk as we did in Sect. 3.1.1 (i.e. to offer the unconditional security to the clients, if we cannot have it for both sides), since the bank can usually choose the signature schemes and the security parameters, thus it can make its own security as large as it wants; and the bank would have to trust cryptographical security in a system based on conventional signatures only, too. A second advantage is that, as soon as a forgery should occur, the bank is sure about it (since it would notice for itself if its own signatures are forged, and for forged signatures of clients, a proof of forgery must be presented); hence the bank can take appropriate action. Thirdly, such a payment system might be advantageous not only for the clients, but also for the bank, since the unconditional security for the clients can be a very good marketing argument.

One can maintain the protocols from previous digital payment systems, except that the bank must publish the prekey K. This is no problem if one uses the hiding scheme based on the hardness of factoring, where no interactive test of the correctness of K is needed (cf. Sect. 2.2.3): Then this publication can easily be incorporated into previous protocols, since at the same time, the bank must publish some other data anyway, e.g. which signature scheme is used, the security parameters, and the bank's public key. All clients can then test if K is indeed odd, before they choose their secret and public keys (just as they test if sensible security parameters have been chosen). The courts carry out the same tests.

The efficiency of the payment system need not be reduced very much: On the one hand, the efficient case with just one risk-bearer could be used. On the other hand, the messages to be signed in a payment system are highly standardized and can therefore be made very short by suitable encoding. A further efficiency improvement is presented in the next section.

3.1.3 Protocols with 1-bit messages

Normal payment system: One can even achieve that the clients sign 1-bit messages only. In this way, we obtain a payment system which is definitely practical. For this, the protocol of how a client gives an order (for a remittance or a withdrawal) is changed a bit (see Fig. 6):

1. First, the client tells the bank what its i-th order will be. This message need not be signed. However, one could require a conventional signature to guard against mischief.[5]
2. The bank signs this order, together with the name or account number of the client and the number i, using the conventional signature scheme.
3. Then the client signs just one bit using his i-th signature.[6] This is defined to mean that the i-th order, as signed by the bank, is correct.

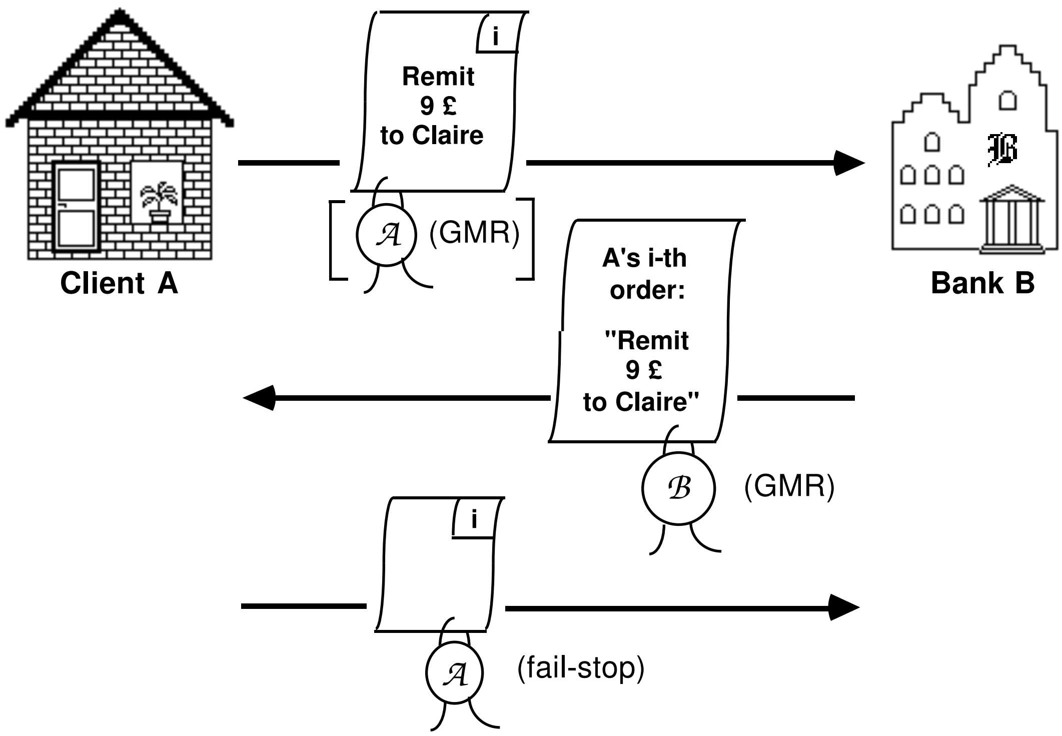

Figure 6: Giving an order in the payment system with 1-bit messages; seals depict signatures.

If the bank receives this signature, it carries out the order. If the client later claims that the bank has carried out the i-th order incorrectly, a court can decide as follows: First the bank must prove that it was allowed to carry out the i-th order, by showing the client's signature

---

5 The mischief would be that someone sends such messages in the name of someone else, although nothing can be gained by that behaviour.

6 Thus only one preimage r_i is needed for the i-th signature, not two preimages $r_{i,0}$ and $r_{i,1}$ as with normal one-time signatures. The whole signature consists of r_i and the tree authentication for pk_i with respect to PK.

from Step 3. If it can do this, and the client cannot present a proof of forgery, the client must show the i-th order with the bank's signature from Step 2. This counts as the valid order.

As in Sect. 3.1.2, the clients are unconditionally secure, and the bank cryptographically: The client only signs the bit (in Step 3) after the bank has signed the correct order in Step 2. She is unconditionally secure against forgeries or her own signature, and the received signature is unconditionally valid. Conversely, a client could ony cheat the bank either by computing a proof of forgery for her own signature, or by forging the bank's signature under an incorrect order. Neither is possible on the cryptographical assumption.[7]

Untraceable payment system: A special payment system where fail-stop signatures can be used without any loss in efficiency (since the clients only sign 1-bit messages anyway) is that from [ChFN_90]. In this system, the payment relations between the clients are unconditionally untraceable (but not balance changes on the account of each client, see, e.g., [PWP_90, BüPf_89]), and it was explicitly regretted that the clients were not unconditionally secure against fraud as well. Using fail-stop signatures, they are.

3.2 Byzantine agreement

An interesting application for fail-stop signatures from a theoretical point of view is Byzantine agreement [PeSL_80], i.e. protocols to achieve reliable broadcast where this is not available physically.

Assume a synchronous network of n processors. Each pair of them can communicate securely. One processor, the sender, is to distribute a message to the others. There are up to t faulty processors, which can behave arbitrarily. Nevertheless, all good processors must agree on the same message (consistency), and if the sender is good, this must be the correct one (correctness).

All previous Byzantine agreement protocols can be classified according to whether they need a cryptographical assumption or not.

- Without a cryptographical assumption, errorless Byzantine agreement can be achieved iff $t < n/3$ [PeSL_80], and with an exponentially small error probability at least for $t < n/2$ [BaPW_91].
- If one assumes that a conventional digital signature scheme is secure, and that the public keys for it have been distributed correctly, Byzantine agreement can be achieved for arbitrary $t \le n$ [PeSL_80].

Using fail-stop signatures, we obtain two types of Byzantine agreement protocols with new security features:

---

[7] A somewhat senseless attack against this payment system, and what would be done against it, should be mentioned: The client could claim that the bank refused to carry out her i-th order, e.g. a withdrawal of an amount x, by not sending the signature in Step 2. The bank, however, would maintain that it had already received a different i-th order, e.g. a withdrawal of a smaller amount y, and that it had already signed that (but not yet received the client's signature in Step 3), so that it could not sign the changed order now.

If such a situation happens seldom, one can just skip the i-th signature and restart the protocol as if the contested order were the $(i+1)$-st one. If it happens often, and since the number of signatures that the client can make with respect to one key is limited, the court should lay down that the i-th order is carried out in the client's version. The court must then store that if the client later shows the signature of the bank under the other version, that will be invalid.

- **Robust fail-stop Byzantine agreement** works for any $t \leq n$ if the signatures cannot be forged; and if the cryptographical assumption turns out to be wrong, and a forgery appears, everybody notices this immediately and can stop participation.
- **Adaptive Byzantine agreement** can only be disturbed if the faulty processors both break the signature scheme and are too many, i.e. $t \geq n/3$ or $n/2$, resp. Thus, whereas previously, one had to decide if one would rather trust a cryptographical assumption or that not too many processors would be faulty, and then select the appropriate protocol, this new protocol handles both cases adaptively.

In [WaPf_89], robust fail-stop Byzantine agreement served to construct a network where the senders and recipients of messages are unconditionally untraceable, and, on a cryptographical assumption, the network cannot be disrupted. This was an unsolved problem from [Chau_88].

4 Summary

We introduced fail-stop signatures, a new kind of digital signature which is more secure than conventional ones, and discussed their advantages (Ch. 1).

As an actual fail-stop signature scheme, we constructed the hiding schemes (Ch. 2). The essential ideas were

- one-time signatures with bundling functions, i.e. cryptographically collision-free functions where the sets of preimages have a guaranteed minimum size,
- tree authentication with cryptographically collision-free hash functions, and
- the rule that in all cases, a collision counts as a proof of forgery.

First a construction from arbitrary claw-free pairs of permutations was shown, then a special case based on the difficulty of factoring. The resulting system is feasible, but not efficient in the general case (cf. Sect. 2.2.4). However, no theoretical reason can be seen why more efficient fail-stop signatures should not exist.

Payment systems are an application where fail-stop signatures are particularly suitable. We demonstrated that by the use of fail-stop signatures, particularly advantageous security features can be obtained (Sect. 3.1.1-2), and the efficiency decreases only minimally, or not at all (Sect. 3.1.3).

Acknowledgements: We thank *Andreas Pfitzmann* for several good ideas, *Gerrit Bleumer*, *Manfred Böttger*, *David Chaum*, *Dirk Fox* and *Jörg Lukat* for helpful discussions and the *DFG* (German Science Foundation) for financial support.

References

BaPW_91 Birgit Baum-Waidner, Birgit Pfitzmann, Michael Waidner: Unconditional Byzantine Agreement with Good Majority; Symp. on Theoretical Aspects of Computer Science (STACS), Hamburg 1991; LNCS, Springer-Verlag, Berlin 1991.

BeMi_88 Mihir Bellare, Silvio Micali: How to sign given any trapdoor function; 20th Symposium on Theory of Computing (STOC) 1988, ACM, New York 1988, 32-42.

Bleu_90 Gerrit Bleumer: Vertrauenswürdige Schlüssel für ein Signatursystem, dessen Brechen beweisbar ist; Studienarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe 1990.

BlPW_90 Gerrit Bleumer, Birgit Pfitzmann, Michael Waidner: A Remark on a Signature Scheme where Forgery can be Proved; Eurocrypt '90 – Abstracts, Århus, May 1990, 403–407.

BrDe_86 Ernest F. Brickell, John M. DeLaurentis: An Attack on a Signature Scheme Proposed by Okamoto and Shiraishi; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 28-32.

BüPf_89 Holger Bürk, Andreas Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

Chau_88 David Chaum: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability; Journal of Cryptology 1/1 (1988) 65-75.

ChDG_88 David Chaum, Ivan B. Damgård, Jeroen van de Graaf: Multiparty Computations ensuring privacy of each party's input and correctness of the result; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 87-119.

ChFN_90 David Chaum, Amos Fiat, Moni Naor: Untraceable Electronic Cash; Crypto '88, LNCS 403, Springer Verlag, Berlin 1990, 319-327.

ChRo_90 David Chaum, Sandra Roijakkers: Unconditionally Secure Digital Signatures; Abstracts of Crypto '90, Santa Barbara 1990, 209-217.

Damg_88 Ivan Bjerre Damgård: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

DaPr_85 Donald W. Davies, Wyn L. Price: Digital Signatures - An Update; Proc. 7th international conference on computer communication (ICCC) Sydney 1984, "The New World of the Information Society", J. M. Bennett, T. Pearcey (eds.); Elsevier Science Publishers B. V. (North-Holland), 1985, 843-847.

Denn_84 Dorothy E. Denning: Digital Signatures with RSA and Other Public-Key Cryptosystems; Communications of the ACM 27/4 (1984) 388-392.

DiHe_76 Whitfield Diffie, Martin E. Hellman: New Directions in Cryptography; IEEE Transactions on Information Theory 22/6 (1976) 644-654.

EAKM_86 Dennis Estes, Leonard M. Adleman, Kireeti Kompella, Kevin S. McCurley, Gary L. Miller: Breaking the Ong-Schnorr-Shamir Signature Scheme for Quadratic Number Fields; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 3-13.

EkHV_88 Jan Ekberg, Siegfried Herda, Jorma Virtamo: TeleTrusT - Technical concepts and basic Mechanisms; Research into Networks and Distributed Applications; R. Speth (ed.) Elsevier Science Publishers B. V., Brussels and Luxembourg, 1988, 523-533.

Elga_85 Taher ElGamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms; IEEE Transactions on Information Theory 31/4 (1985) 469-472.

GiMS_74 E. N. Gilbert, F. J. Mac Williams, N. J. A. Sloane: Codes which detect deception; The Bell System Technical Journal 53/3 (1974) 405-424.

Gira_88 Marc Girault: Hash-functions using modulo-N operations; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 217-226.

GoMR_88 Shafi Goldwasser, Silvio Micali, Ronald L. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

GoMR_89 Shafi Goldwasser, Silvio Micali, Charles Rackoff: The Knowledge Complexity of Interactive Proof Systems; SIAM J. Comput. 18/1 (1989) 186-207.

GoMW_86 Oded Goldreich, Silvio Micali, Avi Wigderson: Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design; 27th

Symposium on Foundations of Computer Science (FOCS) 1986, IEEE Computer Society, 1986, 174-187.

JoCh_86 Wiebren de Jonge, David Chaum: Attacks on Some RSA Signatures; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 18-27.

Merk_80 Ralph C. Merkle: Protocols for Public Key Cryptosystems; Proceedings of the 1980 Symposium on Security and Privacy, April 14-16, 1980 Oakland, California, 122-134.

Merk_82 Ralph C. Merkle: Protocols for Public Key Cryptosystems; AAAS Selected Symposium 69, Secure Communications and Asymmetric Cryptosystems; Gustavus J. Simmons (ed.); Westview Press, Boulder 1982, 73-104.

Merk1_82 Ralph Charles Merkle: Secrecy, authentication, and public key systems; UMI Research Press 1982.

Merk_88 Ralph C. Merkle: A digital signature based on a conventional encryption function; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 369-378.

NaYu_89 Moni Naor, Moti Yung: Universal One–way Hash Functions and their Cryptographic Applications; 21st Symposium on Theory of Computing (STOC) 1989, ACM, New York 1989, 33-43.

Odly_84 Andrew M. Odlyzko: Cryptanalytic Attacks on the Multiplicative Knapsack Cryptosystem and on Shamir's Fast Signature Scheme; IEEE Transactions on Information Theory 30/4 (1984) 594-601.

PeSL_80 Marshall Pease, Robert Shostak, Leslie Lamport: Reaching Agreement in the Presence of Faults; Journal of the ACM 27/2 (1980) 228-234.

Pfit_89 Birgit Pfitzmann: Für den Unterzeichner sichere digitale Signaturen und ihre Anwendung; Diplomarbeit, Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe 1989.

PfWa_90 Birgit Pfitzmann, Michael Waidner: Formal Aspects of Fail-stop Signatures; Interner Bericht 22/90, Fakultät für Informatik, Universität Karlsruhe, Dezember 1990.

PfWa1_91 Birgit Pfitzmann, Michael Waidner: Unbedingte Unbeobachtbarkeit mit kryptographischer Robustheit; appears in: Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte, Springer-Verlag, Heidelberg 1991.

PWP_90 Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Datenschutz und Datensicherung DuD14/5-6 (1990) 243-253, 305-315.

QuCo_82 Jean-Jaques Quisquater, C. Couvreur: Fast Decipherment Algorithm for RSA Public-Key Cryptosystem; Electronics Letters 18/21 (1982) 905-907.

Rabi_78 Michael O. Rabin: Digitalized Signatures; Foundations of Secure Computation, ed. by R.A. DeMillo, D.P. Dobkin, A.K. Jones, R.J. Lipton; Academic Press, N.Y. 1978, 155-166.

RSA_78 R. L. Rivest, A. Shamir, L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126, reprinted: 26/1 (1983) 96-99.

WaPf_89 Michael Waidner, Birgit Pfitzmann: Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks; Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 5/89, März 1989.