HILDESHEIMER
INFORMATIK-
BERICHTE

ISSN 0941-3014

Birgit Pfitzmann

Fail-Stop Signatures Without Trees

16/94 (Juni 1994)

UNIVERSITÄT
HILDESHEIM

Dieser Bericht ist
herausgegeben vom

Institut für Informatik

Postfach 10 13 63
31113 Hildesheim

Fail-Stop Signatures Without Trees

Birgit Pfitzmann

Institut für Informatik, Universität Hildesheim
Marienburger Platz 22, D-31141 Hildesheim, Germany
pfitzb@informatik.uni-hildesheim.de

Abstract

We construct the first fail-stop signature scheme where neither the signature length nor the length of the public key grows as a function of the number of messages that can be signed with one key. The computation needed for signing and testing is reduced similarly. This removes one of the main differences between the complexity of ordinary signature schemes and previous fail-stop signature schemes: In the latter, signatures were branches in an authentication tree, and their length is therefore logarithmic in the size of that tree. Our result also bridges the main gap between known lower and upper bounds on the complexity of fail-stop signature schemes. The construction is based on one-way accumulators.

1 Introduction

Fail-stop signatures, introduced in [WaPf1_90, PfWa2_91], are digital signature schemes with the following additional property: If it ever happens that an adversary with unexpectedly large computing power breaks the system (which cannot be excluded with any of the normal signature schemes) and forges a signature of a participant A, then A can prove that this has happened. Thus any third party who has to decide about the validity of that signature can distinguish unambiguously between the following two cases:

1. The system was broken — and thus A should not be responsible for the signature.
2. A is lying, i.e., she tries to disavow a real signature, or she has leaked her secret key to someone else — and thus A is responsible for the signature.

Of course, it must be infeasible for a polynomially bounded adversary to break the system at all, just as with ordinary digital signature schemes. Detailed formal definitions of the concept are given in [PePf_93].

Efficient fail-stop signature schemes were presented in [HePe_93, HePP_93]. Efficient meant that a signature under one message is of similar length as an ordinary signature (320 bits in [HePe_93] with currently normal security parameters), and the computation needed to compute and verify it is also comparable to ordinary signatures. However, this only holds if just one message is signed. For signing N messages, and assuming that the public key should be short, all known schemes use some form of tree authentication [PfWa2_91], adapted from [Merk_80, Merk_88]. Hence the length of each signature grows logarithmically with the number N of messages that can be signed before a new public key has to be distributed.

It was an open question if signatures could be shorter, since a lower bound from [HePP_93] says that the number of secret random bits needed to sign N messages grows linearly with N. Thus, in a certain sense, the secret key is so long (although it need not exist as a whole all the time). It was unclear if the appropriate part of such a long secret key could be linked to a short public key without something like a tree branch in each signature.

We now use ideas from the one-way accumulators from [BeMa_94] to construct fail-stop signature schemes where both signatures and public keys are short, i.e., their length is independent of N. One-way accumulators are hash functions with a quasi-commutative property. Although they were invented to avoid tree authentication in other scenarios, it is not trivial to apply them to fail-stop signatures, since they are not collision-intractable in the usual sense. This is not necessary in the applications in [BeMa_94], but it would be if they were combined with fail-stop signatures in the simplest way: If an unrestricted adversary forges a signature by breaking the hash function, the signer must obtain a proof of forgery. In previous fail-stop signature schemes, any hash collision is accepted as such a proof. This is only sound if it is computationally hard for the signer to find a collision if no forgery is given.

The unforgeability in our construction relies either on the RSA assumption and the use of a conventional hash function in the place of a random function, or on a stronger RSA-like assumption alone. Note that this is some drawback compared to previous fail-stop signature schemes, which are provably unforgeable if factoring or discrete logarithms are infeasible, or even given an arbitrary one-way permutation [DaPP_94].

2 Definition Sketches

In this section, we sketch the components and security properties of fail-stop signature schemes and one-way accumulators.

2.1 Fail-Stop Signatures

A fail-stop signature scheme has five components: a protocol for generating a pair of a secret and a public key, and algorithms for signing, testing a signature, computing a proof of forgery, and verifying a proof of forgery. The algorithms for signing and computing a proof of forgery have the secret key as an input, the other two algorithms the public key. A signature that passes the test is called *acceptable*, and a proof of forgery that passes the verification is called *valid*.

Key generation is a protocol, and not just an algorithm, so that the signer cannot construct false keys where she can construct false proofs of forgery. For simplicity in the notation, we assume that a center trusted by all recipients carries out this protocol with the signer. The simplest type of key generation protocol consists of two steps: First, the center generates a value called a *prekey*, which can be used for all signers. Then each signer constructs her actual secret and public key based on that prekey. The following construction is of this type.

The components of a fail-stop signature scheme with prekey must have three properties:

- *Acceptability of signatures*: If key generation has been carried out successfully, correct signatures must be acceptable.
- *Security for the recipient*: It is infeasible for a polynomially bounded signer to produce an acceptable signature and a valid proof of forgery for it.
- *Security for the signer*: It is impossible for a forger, even with unlimited computing power, to produce an acceptable signature that the signer cannot prove to be a forgery.

The security for the recipient is determined by a security parameter k and the security of the signer by another security parameter σ, because with an exponentially small probability $2^{-\sigma}$, the forger may hit the correct signature. Note that the properties required above imply that forging is infeasible. More details can be found in [PePf_93].

2.2 One-Way Accumulators

One-way accumulators are a type of hash functions introduced in [BeMa_94]. Such a function h hashes two messages of roughly equal size to a result of the length of just one message. More precisely, it is a function

$$h: X \times Y \to X$$

for some sets X and Y. The distinguishing quasi-commutative property is defined as follows: For all $x \in X$ and $y_1, y_2 \in Y$,

$$h(h(x, y_1), y_2) = h(h(x, y_2), y_1).$$

The one-way property, in this case, means that it is infeasible to find a collision if x, y, and y' are given, i.e., to find an x' such that $h(x, y) = h(x', y')$. This is more related to the definition of universal one-way hash functions [NaYu_89] than that of collision-intractable hash functions [Damg_88].

More precisely, a family of one-way accumulators is defined. It contains the following algorithms (a few more than in [BeMa_94], since they occur in the constructions and applications in that abstract): a key generation algorithm — the function selected by a key n is denoted by $h_n: X_n \times Y_n \to X_n$ — and algorithms to evaluate h on input (n, x, y), and to select random elements from X_n and Y_n. The one-way property is then defined over random choices of n, x, y, and y'.

The special stronger form of collision-intractability that we need is discussed in Section 4.

3 The Constructions

Our constructions rely on an arbitrary one-time fail-stop signature scheme (i.e., a scheme for signing just one message, like [HePe_93]), and the specific family of one-way accumulators constructed in [BeMa_94]. In this family, a key is a specific RSA modulus n, and

$$h_n(x, y) := x^y \bmod n.$$

Prekey Generation. The center chooses

- a prekey from the underlying one-time fail-stop signature scheme,
- a collision-intractable hash function Ω (possibly with additional properties to be discussed below), and
- two integers

$$n, x,$$

where n is a rigid RSA modulus as defined in [BeMa_94] and x is a random number modulo n. The former means that $n = pq$ with $p = 2p'+1$ and $q = 2q'+1$, where p, p', q, q' are prime and p and q are of equal length; this was required so that $\mathbb{Z}_n^*$ has $4p'q'$ elements and thus almost none of its elements lie in subgroups smaller than QR_n. We have maintained this requirement, although it is not strictly necessary in the following, but omitted that x should be a quadratic residue.

Main Key Generation. Essentially, the signer chooses key pairs (sk_i, pk_i) from the underlying one-time fail-stop signature scheme for $i = 1, \ldots, N$ (which will be used to sign the N individual messages, similar to schemes with bottom-up tree authentication), and all the "public" keys pk_i are accumulated to yield the real public key pk. However, not all values pk_i are immediately suitable for accumulation if we want to guarantee a form of collision-intractability. In particular, the values accumulated must not consist of too small prime factors. Let *unsmooth* denote the exact predicate that is required. The simplest definition of *unsmooth* for the security considerations is

$$unsmooth := prime.$$

The following particular steps are proposed (alone or combined, see Section 4). For each i:

1. The collision-intractable hash function Ω is applied. This makes the inputs to accumulation short enough, and Ω can also serve as an approximation to a random oracle. (Then, of course, one has to require that no evidence to the contrary is known about Ω.)
2. Now one searches for a value r_i that fulfils *unsmooth* among the numbers that start with $\Omega(pk_i)$ and have a few bits appended, say l bits with $l \approx \text{loglog}(n)$. Thus

   $$r_i = 2^l \Omega(pk_i) + dist_i$$

   for some $dist_i \in \{0, \ldots, 2^l-1\}$.
3. If no unsmooth value r_i is found in this range, the choice of (sk_i, pk_i) is repeated.

The tests for *unsmooth* make key generation somewhat slower than in schemes with tree authentication, but the factor is independent of N. Note that the simpler-looking scheme where only $r_i = \Omega(pk_i)$ is allowed is the special case $l = 0$, but $l \approx \text{loglog}(n)$ is more efficient, since one usually does not have to repeat the one-time key generation.

Now the r_i's are accumulated. The public key is

$$pk := x^{r_1 \cdots r_N} \bmod n.$$

For each i, the signer also stores the values $dist_i$ and

$$accu_i := x^{r^*_i} \bmod n,$$

where r^*_i is the product of all the r_j's, except for r_i. (The list of all values $accu_i$ can be computed with $N \log_2(N)$ exponentiations with a tree-like evaluation, and this can be done off-line after pk has been published.)

Signing. The signature on the i-th message, m_i, is

$$s := (s_i, pk_i, dist_i, accu_i),$$

where s_i is the one-time signature on m_i with the one-time key sk_i, and pk_i, $dist_i$, and $accu_i$ are needed to authenticate this signature with respect to the accumulated public key pk.

Testing. Given the public key pk, a value $s = (s_i, pk_i, dist_i, accu_i)$ is an acceptable signature on a message m iff

- s_i is an acceptable one-time signature on m with respect to pk_i,
- pk_i fulfils the precondition for accumulation, i.e., $r_i := 2^l\Omega(pk_i) + dist_i$ fulfils the predicate *unsmooth*, and
- the authentication of pk_i with respect to pk is correct, i.e.,

$$accu_i^{r_i} = pk \bmod n.$$

Proving Forgeries. If a signer is presented with a forged acceptable signature (s', pk', $dist'$, $accu'$) on a message m, she first searches if pk' is any of her correct one-time "public" keys pk_i. (This may take some time, but since proving a forgery only occurs in extreme cases, it is tolerable. Similarly, it does not matter that proofs of forgery are very long.)

- If $pk' = pk_i$, she tries to prove s' to be a forgery with respect to pk_i in the one-time fail-stop signature scheme.
- Otherwise, she shows the accumulator collision

$$(((pk_1, dist_1), \ldots, (pk_N, dist_N)), (pk', dist', accu')).$$

Verifying Proofs of Forgery. If the signer claims to have a proof of forgery of the one-time fail-stop signature scheme, one verifies that. Otherwise, one verifies the accumulator collision $(((pk_1, dist_1), \ldots, (pk_N, dist_N)), (pk', dist', accu'))$ as follows: Compute $r_i := 2^l\Omega(pk_i) + dist_i$ for $i = 1, \ldots, N$ and $r' := 2^l\Omega(pk') + dist'$. Verify if

- all the values r_i and r' fulfil the predicate *unsmooth*,
- pk' does not occur among the values pk_i, and
- $$x^{r_1 \cdots r_N} = accu'^{r'} \bmod n.$$

4 Security

Before considering the real security properties, we have to show that everything works in the correct case.

4.1 Acceptability of Signatures

It is easy to see that all correctly generated signatures are acceptable; in particular, the accumulation condition is fulfilled since

$$accu_i^{r_i} = x^{r^*_i r_i} = x^{r_1 \cdots r_N} = pk \bmod n.$$

4.2 Effectiveness of Key Generation

We have to show that key generation is polynomial-time. It suffices to show that the values tested for unsmoothness fulfil this predicate with roughly the same probability as random numbers.

First consider the non-optimized case $l = 0$. Then our claim is clearly fulfilled if Ω is considered as a random function. Otherwise one can first try to prove it for the given individual one-time signature scheme and Ω. E.g., it holds for the scheme from [HePe_93] and $\Omega = id$, since public keys can easily be seen to be uniformly distributed, and one can also use collision-intractable hash functions that are secure on the same assumption (cf. [PePf_93]), since these particular hash functions map random inputs to random outputs. In the general case, one may have to randomize after the application of Ω (although, with sufficiently large l and some assumptions about the distribution of primes, probably not). For instance, one can apply a specific compression function for the range of Ω, or a pseudo-random permutation chosen and published by the signer [GoGM_86]. (It does not matter that pseudo-random permutations only look random as long as their description is unpublished: If their outputs would not be prime with the same probability as random numbers, they could have been distinguished even without being published.)

If $l \neq 0$, one tests several values from certain intervals. In practice, this does not make much difference (just as prime numbers are often generated by step-wise search). In theory, one can see that choosing just one value $dist_i$ randomly yields one real random number (or, depending on Ω, something similar), and that testing more numbers only yields a polynomial overhead as long as $l \approx \mathrm{loglog}(n)$), as proposed in Section 3.

4.3 Security for the Signer

Security for the signer can be proved quite easily: Any successful forgery, i.e., any acceptable signature on a message that the signer had not signed, belongs to one of two cases:

1. It contains a one-time forgery, i.e., a forged signature with respect to one of the real values pk_i.
2. It contains a value *pk'* different from the real pk_i's.

In Case 1, the forgery is provable with the same probability as in the underlying one-time signature scheme. (It does not matter that the choice of pk_i may have been repeated: The probability that a forger guesses exactly the correct signature is small for each pk_i individually.) In Case 2, the signer obtains an accumulator collision without any error probability, and one can easily see that it passes the verification.

4.4 Security for the Recipient

By the security for the recipient in the underlying one-time fail-stop signature scheme, it is infeasible to compute a proof of forgery in that scheme (unless a forgery is given, of course). Thus it remains to show that it is infeasible to construct accumulator collisions of the type that are valid proofs of forgery. Note that the algorithm $\tilde{A}$ that tries to find such a collision is given n and x, but that it can choose all the other parameters itself, in contrast to the one-way property from [BeMa_94] described in Section 2.2.

Any accumulator collision $(((pk_1, dist_1), \ldots, (pk_N, dist_N)), (pk', dist', accu'))$ belongs to one of three types. (As before, $r' := 2^l\Omega(pk') + dist'$ and $r_i' := 2^l\Omega(pk_i) + dist_i$.)

1. $r' \in \{r_1, \ldots, r_N\}$. Then there is an Ω-collision: $2^l\Omega(pk') + dist' = 2^l\Omega(pk_i) + dist_i$ implies $\Omega(pk') = \Omega(pk_i)$ (this was the reason for appending $dist_i$, instead of simply adding it to $\Omega(pk_i)$) and $pk' \neq pk_i$ is a precondition for an accumulator collision.
2. $r' \notin \{r_1, \ldots, r_N\}$, but r' divides the product $r_1 \cdots r_N$ in $\mathbb{N}$.
3. r' does not divide the product $r_1 \cdots r_N$ in $\mathbb{N}$.

If $\tilde{A}$ is successful with non-negligible probability, at least one of these types occurs with non-negligible probability, too.

Since Ω was assumed to be collision-intractable, this cannot be Type 1.

If we define the predicate *unsmooth* as *prime*, it cannot be Type 2 either. (With a weaker *unsmooth*, one has to consider probabilities, as sketched in [BeMa_94, end of Section 4.2].)

Thus we can concentrate on showing that it is infeasible to construct collisions of Type 3. We consider this first without and then with a "random oracle assumption" for Ω. This assumption means that Ω is replaced by a function chosen randomly among all functions with the appropriate domain and codomain. It was first used for the signature scheme in [FiSh_87]; recently, the approach was formalized more generally in [BeRo_94]. The reason for working with and without it is that it is very helpful, but not a cryptographic assumption in the usual sense, just an analogy: The nearest cryptographic definition is that of pseudo-random function

families. However, only one group of people can evaluate a pseudo-random function, and it looks random for another group of people; with a random oracle, both properties hold for all people.

Without a Random Oracle Assumption. In this subsection, the only assumption about Ω is collision-intractability. Note that this is also fulfilled if Ω is the identity, i.e., no hashing is used at all; but then one either has to use a rather large value n or split each pk_i into two or more blocks. We only consider this case in combination with *unsmooth* defined as *prime* — otherwise an adversary might construct collisions of Type 2 by multiplying small primes. (Note that $r't = r_1 \cdots r_N$ yields a collision with $accu := x^t$.)

We show that the construction is secure on the following assumption.

Assumption 1 (Very strong RSA assumption). For all probabilistic polynomial-time algorithms $\tilde{A}_1$, all $c > 0$, and all sufficiently large k,

$$k^{-c} \geq \mathrm{P}(y^e = x \bmod n \wedge e \text{ prime} \mid n \in_R RSA_Mod(k); x \in_R \mathbb{Z}_n; (y, e) \leftarrow \tilde{A}_1(n, x)),$$

where "$\in_R$" means random choice from a set, and $RSA_Mod(k)$ is the set of rigid RSA moduli of length k. ♦

Thus the adversary $\tilde{A}_1$ is given n and x as in a usual RSA assumption, but he may choose the "public" exponent e for which he extracts the root. We are neither aware that this assumption has been used before or of any corroboration that it should be hard, nor can we break it. Four obvious attacks do not work, i.e., they are equivalent to breaking some other problem widely believed to be hard:

- If the adversary chooses a random e first, he has to break RSA.
- If he chooses a random y first, he has to compute a discrete logarithm.
- If he tries to compute y as some power $x^d \bmod n$ together with e, then $x^{de} = x \bmod n$, and for almost all values of x this is known to yield a factorization of n.
- The attacker could try to choose special values e for which RSA would be easier to break. However, no such exponents seem to be known. (There are attacks for short *secret* exponents [Wien_90], but our e corresponds to the public exponent, and we see no way for the attacker to influence the corresponding secret exponent. A well-known attack on short public exponents (cf. [Håst_86]) only applies to situations where the attacker sees three messages encrypted with that exponent using different moduli.)

Theorem 1. The construction from Section 3 with a collision-intractable function (family) Ω and with *unsmooth* = *prime* is secure on Assumption 1. ♦

Proof sketch. It is obviously sufficient to prove the following formula (where the adversary may choose r' and the r_i's freely without the pk's): For all probabilistic polynomial-time algorithms $\tilde{A}$, all $c > 0$, and all sufficiently large k,

$$k^{-c} \geq \mathrm{P}(accu^{r'} = x^{r_1 \cdots r_N} \bmod n \wedge r' \notin \{r_1, \ldots, r_N\} \wedge r', r_1, \ldots, r_N \text{ prime} \mid n \in_R RSA_Mod(k); x \in_R \mathbb{Z}_n; (accu, r', r_1, \ldots, r_N) \leftarrow \tilde{A}(n, x)).$$

If this were not true, we would obtain a contradiction to Assumption 1: Let $e := r'$ and $r := r_1 \cdots r_N$. Now the e-th root y of x can be constructed as in [Sham_83, BeMa_94]: Compute $a, b \in \mathbb{Z}$ with $ar + br' = 1$ with the extended Euclidean algorithm and let $y := accu^a x^b$. Thus $y^e = accu^{ar'} x^{br'} = x^{ra+br'} = x \bmod n$. □

With a Random Oracle Assumption. With a random oracle assumption in the place of Ω, and still *unsmooth* = *prime*, security can be shown on the normal RSA assumption.

Assumption 2 (Normal RSA assumption). For all probabilistic polynomial-time algorithms $\tilde{A}_2$, all $c > 0$, and all sufficiently large k,

$$k^{-c} \geq \mathrm{P}(y^e = x \bmod n \mid n \in_R RSA_Mod(k); x \in_R \mathbb{Z}_n; e \in_R \mathbb{Z}_{\phi(n)}^*; y \leftarrow \tilde{A}_2(n, x, e)). \quad ♦$$

Note that there is another possible "normal RSA assumption", which is often not distinguished precisely: One can fix some e, such as 3, and vary just n and x.

Theorem 2. The construction from Section 3 with a random oracle Ω and with *unsmooth* = *prime* is secure on Assumption 2. ♦

Proof sketch. Let $\mathcal{F}$ be the set of functions with the domain and codomain as required for Ω. We have to show for all probabilistic polynomial-time oracle algorithms $\tilde{A}$, all $c > 0$, and all sufficiently large k,

$$k^{-c} \geq \mathrm{P}(accu^{r'} = x^{r_1 \cdots r_N} \bmod n \wedge r' \notin \{r_1, \ldots, r_N\} \wedge r', r_1, \ldots, r_N \text{ prime} \mid n \in_R RSA_Mod(k); x \in_R \mathbb{Z}_n; \Omega \in_R \mathcal{F}; (accu, (pk', dist'), (pk_1, dist_1), \ldots, (pk_N, dist_N)) \leftarrow \tilde{A}^{\Omega}(n, x); r' := 2^l \Omega(pk') + dist'; r_i := 2^l \Omega(pk_i) + dist_i \text{ for } i := 1, \ldots, N),$$

where $\tilde{A}^{\Omega}$ means $\tilde{A}$ with access to the oracle Ω. Just as in the previous subsection, if $\tilde{A}$ is successful, one can also compute the r'-th root of x. Hence it is sufficient to prove (with the same quantification)

$$k^{-c} \geq \mathrm{P}(accu^{r'} = x \bmod n \wedge r' \text{ prime} \mid n \in_R RSA_Mod(k); x \in_R \mathbb{Z}_n; \Omega \in_R \mathcal{F}; (accu, pk', dist') \leftarrow \tilde{A}^{\Omega}(n, x); r' := 2^l \Omega(pk') + dist').$$

Without loss of generality, we can assume that $\tilde{A}$ has asked the oracle for $\Omega(pk')$. The number of values that $\tilde{A}$ asks for is bounded by a polynomial $Q(k)$. Whatever strategy $\tilde{A}$ uses, it amounts to the same thing as if it were given a list of $Q(k)$ random numbers ρ and had to select r' among the numbers $2^l \rho + dist$. (To see this, recall that a random oracle is equivalent to the following machine: whenever asked for $\Omega(pk)$ for a new value pk, it generates a random number r as an answer, and it stores all previous pairs (pk, r) so that it answers with the same r

if asked the same *pk* twice.) Thus this new adversary $\tilde{A}^*$ is given a list of exponents and has to extract a root for at least one of them. If this were possible with non-negligible probability, it would also be possible — with a probability smaller by the factor $Q(k)\bullet 2^l$, where 2^l is polynomial in k — to extract an r'-th root for one given random value r'. (For this, choose $Q(k)$ – 1 more random numbers ρ, make a list of intervals as above, put r' into a random place in this list, and call $\tilde{A}^*$.) This would contradict Assumption 2 with $e = r'$. □

If one relaxes *unsmooth* (to speed up key generation), and one finds the probability that the adversary can find a collision of Type 2 among the random numbers provided by the oracle acceptably small (which is a bit risky since one has to guess how fast he can evaluate Ω), one cannot necessarily compute the r'-th root of x from $accu^{r'} = x^{r_1 \cdots r_N}$. However, one obtains the g-th root, where $g := r'/gcd(r', r_1 \cdots r_N)$. Hence one can prove the security on another RSA-like assumption; the difference to Assumption 2 is that the adversary has also won if he manages to extract a root for any divisor $g \neq 1$ of the given e.

5 Conclusion

We have used the principles of one-way accumulators to construct fail-stop signature schemes where the length of both signatures and public keys is constant as a function of the number N of messages that can be signed. With 660-bit moduli n and $l = 10$, and the one-time fail-stop signature scheme from [HePe_93] with 160-bit q and 660-bit p, the length of a signature is 2310 bits; 1990 of them can be sent before the message is known. Signing requires only two multiplications, testing about 4 exponentiations (assuming that one exponentiation suffices in the primality test [DaLP_93]). Key generation, however, takes significantly longer than in constructions with trees. The security is based either on the normal RSA assumption and the use of a conventional hash function in the place of a random function, or solely on a stronger RSA assumption that we encourage the reader to cryptanalyse.

To avoid the precomputation of a very long secret key, one can combine the constructions with top-down tree authentication. For instance, one might use accumulation for 1024 pairs (sk_i, pk_i) each, form a tree with 2 levels of such structures, and sign the "public" keys of the lower level with the secret keys of the upper level on demand.

Acknowledgment

Thanks to *Michael Waidner*, *Joachim Biskup*, and *Andreas Pfitzmann* for helpful comments on this paper. An anonymous referee of a previous version drew my attention to [Wien_90], and made me learn never to give what's now *e* the name *d* again.

References

BeMa_94 Josh Benaloh, Michael de Mare: One-Way Accumulators: A Decentralized Alternative to Digital Signatures; Eurocrypt '93, LNCS 765, Springer-Verlag, Berlin 1994, 274-285.

BeRo_94 Mihir Bellare, Phillip Rogaway: Entity Authentication and Key Distribution; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 232-249.

DaLP_93 Ivan Damgård, Peter Landrock, Carl Pomerance: Average Case Error Estimates for the Strong Probable Prime Test; Mathematics of Computation 61/203 (1993) 177-194.

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

DaPP_94 Ivan Bjerre Damgård, Torben Pryds Pedersen, Birgit Pfitzmann: On the Existence of Statistically Hiding Bit Commitment Schemes and Fail-Stop Signatures; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 250-265.

FiSh_87 Amos Fiat, Adi Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 186-194.

GoGM_86 Oded Goldreich, Shafi Goldwasser, Silvio Micali: How to construct random functions; Journal of the ACM 33/4 (1986) 792-807.

Håst_86 Johan Håstad: On Using RSA with low exponent in a public key network; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 403-408.

HePe_93 Eugène van Heyst, Torben Pryds Pedersen: How to make efficient Fail-stop signatures; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 366-377.

HePP_93 Eugène van Heijst, Torben Pryds Pedersen, Birgit Pfitzmann: New Constructions of Fail-Stop Signatures and Lower Bounds; Crypto '92, LNCS 740, Springer-Verlag, Berlin 1993, 15-30.

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

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.

PePf_93 Torben Pryds Pedersen, Birgit Pfitzmann: Fail-Stop Signatures; Version 25.2.1993, submitted for publication.

PfWa2_91 Birgit Pfitzmann, Michael Waidner: Fail-stop Signatures and their Application; Securicom 91; 9th Worldwide Congress on Computer and Communications Security and Protection, Paris, 19.-22. March 1991, 145-160.

Sham_83 Adi Shamir: On the Generation of Cryptographically Strong Pseudorandom Sequences; ACM Transactions on Computer Systems 1/1 (1983) 38-44.

WaPf1_90 Michael Waidner, Birgit Pfitzmann: The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 690.

Wien_90 Michael J. Wiener: Cryptanalysis of Short RSA Secret Exponents; IEEE Transaction on Information Theory 36/3 (1990) 553-558.