To appear in Proc. Eurocrypt '97, LNCS, Springer-Verlag, Berlin 1997

Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees*

Niko Barić[1] and Birgit Pfitzmann[2]

[1] dvg Hannover, Postfach 91 02 40, D-30422 Hannover, Germany
[2] Universität Dortmund, Informatik 6, D-44221 Dortmund, Germany;
email pfitzb@ls6.informatik.uni-dortmund.de

Abstract. One-way accumulators, introduced by Benaloh and de Mare, can be used to accumulate a large number of values into a single one, which can then be used to authenticate every input value without the need to transmit the others. However, the one-way property does is not sufficient for all applications.
In this paper, we generalize the definition of accumulators and define and construct a collision-free subtype. As an application, we construct a fail-stop signature scheme in which many one-time public keys are accumulated into one short public key. In contrast to previous constructions with tree authentication, the length of both this public key and the signatures can be independent of the number of messages that can be signed.

1 Introduction

The security of digital signature schemes depends on so-called computational assumptions, e.g., the factoring assumption. If somebody can break the assumption on which the system is based, and if he can therefore get the private key of the signer, he can construct signatures on messages chosen by himself. The signer cannot prove that she did not sign those messages herself.

This disadvantage was overcome with the introduction of "fail-stop" signature schemes, e.g., [WaPf90, PfWa90, HePe93, PePf97]. With these schemes, the signer can produce a so-called proof of forgery to demonstrate that she did not sign a message. This proof shows that the computational assumption has been broken (fail) and that the system should therefore not be used any longer (stop).

Most of the currently known basic constructions of fail-stop signature schemes (FSS schemes) can only be used to sign one single message. FSS schemes for more than one message have been constructed based on these one-time FSS schemes by using tree authentication to authenticate the public one-time keys. Consequently, the length of signatures in such a scheme grows logarithmically in the number of messages that can be signed. The question whether this can be

* Work done while both authors were at the University of Hildesheim. Supported by the DFG (German Research Foundation). A preliminary version was available as [Pfit94], more details can be found in [Bari96].

avoided was also the main gap between known lower and upper bounds on the complexity of fail-stop signature schemes [HePP93].

The accumulators presented in [BeMa94] seem to be a solution to this problem: a large number of values is accumulated into one value z. Later on, for authentication of one of those values, y, an additional value is computed that will authenticate y with respect to z. The length of z and the additional value can be independent of the number of values to be accumulated. If we use this for FSS schemes and accumulate all the public one-time keys, the length of the resulting public key and the signatures can be independent of the number of messages.

The accumulators defined in [BeMa94] have only a one-way property, i.e., given an output, it is hard to find a suitable input. Unfortunately, this is not enough for an FSS scheme, because the adversary may be able to choose the one-time public keys (i.e., the values to be accumulated), and thus to some extent the accumulated output, himself. Therefore we define and construct collision-free accumulators. We take the opportunity to generalize the accumulators defined in [BeMa94] to contain only those properties that are needed for our purpose and also to include newer accumulators from [Nybe96a, Nybe96b]. The new collision-free accumulators are then included into a modular FSS scheme. Thus now we really have a scheme where the length of both the public key and the signatures is independent of the number of messages.

The goal of constructing schemes without trees is similar to recent efforts with non-fail-stop provably secure signatures to shorten the signatures by flat trees [DwNa94, CrDa96], but the measures developed there cannot be used for FSS schemes.

1.1 Organization of this Paper

In Section 2, we present our definitions and constructions of accumulators. In Section 3, we describe conversion algorithms as an interface between the one-time FSS scheme and the accumulator which is used to authenticate the individual public one-time keys. The general construction of our accumulator FSS scheme is given in Section 4. Two example accumulator FSS schemes follow in Section 5.

2 Accumulators

Accumulators were introduced in [BeMa94] as a new way of "summarizing" a large number N of values in one value. The accumulators as defined in [BeMa94] have some properties we do not need for our purposes, so we generalize their definition a little. Then we define some subtypes of accumulators with different levels of security. Nevertheless, the accumulators as given in [BeMa94] are an important subtype that we call *elementary accumulators* (see Section 2.2).

2.1 General Accumulators

Definition 1. A *family a of accumulators* has the following components:

- Sets $accu\_keys(k, N)$, which contain all possible keys for the security parameter k and the number N of values to be accumulated, and a probabilistic polynomial-time algorithm $accu\_gen(k, N)$ that chooses an accumulator key n from $accu\_keys(k, N)$. If the choice is uniformly random, we often simply write $\in_R$.
  In our examples, $accu\_keys(k, N)$ is independent of N.
- Sets Y_n containing the suitable inputs for an accumulator key n.
- A probabilistic polynomial-time algorithm *accu_eval* which, on input an accumulator key n and N values $y_1, \ldots, y_N \in Y_n$, outputs a value z and an auxiliary value aux, which will be used by the other algorithms.
  We write $a_n(y_1, \ldots, y_N)$ instead of $accu\_eval(n, y_1, \ldots, y_N)$.
  Every execution of *accu_eval* with the same input $(n, y_1, \ldots, y_N)$ must yield the same output z.
- A probabilistic polynomial-time algorithm *auth* that, on input n, y_i, and aux, computes a value $accu_i$ from a set $Accu_n$, which is needed to authenticate y_i.
  We write $auth_n(y_i, aux)$ instead of $auth(n, y_i, aux)$.
- A polynomial-time algorithm *authentic* which, on input $(n, z, y_i, accu_i)$, checks whether $y_i \in Y_n$ together with $accu_i \in Accu_n$ is authenticated by z. If so, the output is *ok*, otherwise *not_ok*.
  We write $authentic_n(z, y_i, accu_i)$ instead of $authentic(n, z, y_i, accu_i)$.

Additionally, there must be two polynomial-time algorithms: one that, on input n and y, checks whether $y \in Y_n$, and one that, on input n and $accu$, checks whether $accu \in Accu_n$. Finally, we require that every y_i in the input of a_n can be authenticated by the output of a_n, formally:

$$\begin{aligned}
&\forall k\, \forall N\, \forall n \in accu_keys(k, N)\, \forall (y_1, \ldots, y_N) \in Y_n^N: \\
&\quad \text{If } (z, aux) \leftarrow a_n(y_1, \ldots, y_N) \\
&\quad \text{then } \forall i \in \{1, \ldots, N\}: \\
&\qquad authentic_n\big(z, y_i, auth_n(y_i, aux)\big) = ok.
\end{aligned}$$

In [BeMa94], a one-way property is defined for accumulators. Generalized to our definition, it means that it is hard for an adversary who is given values $(y_1, \ldots, y_N)$, their accumulation result z, and another value y' to find a value $accu'$ that authenticates y' with respect to z.

That article also informally considers a slightly stronger property that we call *strongly one-way*. It means that given only $(y_1, \ldots, y_N)$ and z, it is hard to find a pair $(y', accu')$ such that $authentic_n\big(z, y', accu'\big) = ok$ with $y' \notin \{y_1, \ldots, y_N\}$. I.e., now the attacker can choose the value y' himself. The importance of a strong one-way property was also recognized in [Nybe96a].

For our accumulator FSS scheme, we need an even stronger property, because the adversary might be able to choose all the public one-time keys that are to be accumulated, i.e., not even the values $y_1, \ldots, y_N$ are now given.

Definition 2. A family a of accumulators is *N-times collision-free* for $N \geq 1$ if it is hard to find $y_1, \ldots, y_N$, another value y' and $accu'$ such that y' is authenticated by $accu'$ and $a_n(y_1, \ldots, y_N)$: for all probabilistic polynomial-time adversaries $\widetilde{A}$, all $c > 0$, and all sufficiently large k:

$$\begin{aligned} \mathrm{P}\Big(& authentic_n(z, y', accu') = ok \ \wedge \ y' \notin \{y_1, \ldots, y_N\} \\ & \qquad \wedge\, y', y_1, \ldots, y_N \in Y_n \ \wedge \ accu' \in Accu_n :: \\ & n \leftarrow accu\_gen(k, N); \\ & (accu', y', y_1, \ldots, y_N) \leftarrow \widetilde{A}(k, N, n); \\ & (z, aux) \leftarrow a_n(y_1, \ldots, y_N)\Big) \leq \frac{1}{k^c}. \end{aligned}$$

Definition 3. A family a of accumulators is *collision-free* if a is N-times collision-free for all $N \geq 1$.

2.2 Elementary Accumulators

In [BeMa94], accumulators were defined as functions h_n: $X_n \times Y_n \to X_n$, where n is again an accumulator key. With repeated use as in

$$z = h_n\Big(\cdots h_n\big(h_n(x, y_1), y_2\big), \cdots y_N\Big),$$

where the result of one application of h_n is inserted as the first argument in the next application of h_n, all $y_1, \ldots, y_N \in Y_N$ are accumulated to a value $z \in X_n$ given an initial value $x \in Start_n \subseteq X_n$.

With such a function h_n, we can create an accumulator $a_{(n,x)}$ according to the general definition, where the initial value x is part of the key of a, as follows:

$$(z, aux) = a_{(n,x)}(y_1, \ldots, y_N)$$

with z as above and $aux = (x, y_1, \ldots, y_N)$. We use $(x, y_1, \ldots, y_N)$ as the auxiliary output, so that we can use it for the computation of the values $accu_i$.

In [BeMa94], such a function h_n has to be *quasi-commutative*, i.e.,

$$h_n\big(h_n(x, y_1), y_2\big) = h_n\big(h_n(x, y_2), y_1\big) \text{ for all } x \in X_n \text{ and } y_1, y_2 \in Y_n.$$

We do not need this property for our accumulator FSS scheme, but if one has a function with this property, one can easily construct algorithms to create and verify the values $accu_i$ [BeMa94]:

$$auth_n(y_i, (x, y_1, \ldots, y_N)) = h_n\Big(\ldots h_n\Big(h_n\big(\ldots h_n(x, y_1), \ldots y_{i-1}\big), y_{i+1}\Big), \ldots y_N\Big)$$

and

$$authentic_n(z, y_i, accu_i) = ok \text{ iff } z = h_n(accu_i, y_i).$$

In this case, the list of all values $accu_i$ can be computed with $\mathcal{O}(N \cdot \log_2 N)$ applications of h_n with a tree-like evaluation. This can be done offline after z has been published.

2.3 Examples

In the following two subsections, we give two examples of accumulators. Both are based on the elementary accumulator given in [BeMa94], but with some modifications to fulfill the collision-freeness needed for the accumulator FSS scheme.

Another elementary and strongly one-way accumulator is described in [Nybe96a, Nybe96b]. In short, it uses a hash function h that generates a long random output o_i of fixed length $r \cdot d$ for every input y_i, where r and d are two security parameters. Then o_i is transformed into a bitstring b_i of length r that has far more 1's than 0's. To accumulate the values $(y_1, \ldots, y_N)$, the corresponding strings b_i are multiplied modulo 2 coordinatewise. In the result, a bit can be zero only if at least one b_i has a zero bit at the same place. The main advantage of this accumulator is the absence of any trapdoor information, whereas in the following accumulators based on the RSA assumption, someone knows the factors of the RSA modulus. A disadvantage is its long output, too long for the public key of an FSS scheme.

RSA Accumulator Without Random Oracle. The first example is almost the same accumulator as presented in [BeMa94], based on the elementary accumulator function $h_n(x, y) = x^y \bmod n$.

Definition 4. The following family a^{RSA} is called *RSA accumulator without random oracle*:

- $\mathit{accu\_keys}^{\mathrm{RSA}}(k, N) := \big\{(n, x) \,\big|\, n \in \mathit{RSA\_Mod}(k) \wedge x \in \mathbb{Z}_n\big\}$
- $Y^{\mathrm{RSA}}_{(n,x)} := \{y \mid y < n \wedge y \text{ prime}\}$
- $a^{\mathrm{RSA}}_{(n,x)}(y_1, \ldots, y_N) := x^{y_1 \cdots y_N} \bmod n$
- $\mathit{auth}^{\mathrm{RSA}}_{(n,x)}\big(y_i, (x, y_1, \ldots, y_N)\big) := x^{y_1 \cdots y_{i-1} y_{i+1} \cdots y_N} \bmod n$
- $\mathit{authentic}^{\mathrm{RSA}}_{(n,x)}(z, y, \mathit{accu}) := \mathit{ok}$ iff $\mathit{accu}^y \equiv z (\mathrm{mod}\, n)$

Here, *RSA_Mod*(k) is the set of RSA moduli of length k [RSA78]. The difference to the original accumulator is the restriction of the input domain to prime numbers. In addition, to prove collision-freeness, we have to make a stronger RSA assumption.

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

$$\begin{aligned} &\mathrm{P}\big(y^e \equiv x (\mathrm{mod}\, n) \wedge e \text{ prime} \wedge e < n :: \\ &\quad n \in_R \mathit{RSA\_Mod}(k); x \in_R \mathbb{Z}_n; (y, e) \leftarrow \widetilde{A}(n, x)\big) \le k^{-c}. \end{aligned}$$

Thus the adversary $\widetilde{A}$ is given n and x as in a usual RSA assumption, but he may choose the exponent e for which he extracts the root. We are neither aware 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 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 find d and e with $y = x^d$ and $(x^d)^e = x$, then $\mathrm{ord}(x)$ divides $f := de - 1$, where $\mathrm{ord}(x)$ is the smallest $i > 0$ with $x^i \equiv 1 \pmod{n}$. He can then also break RSA for the same n and x: Let a random public exponent e' be given. It is sufficient to consider the case where e' is prime and no factor of f. Then we set $d' := e'^{-1} \bmod f$ and obtain $(x^{d'})^{e'} \equiv x \pmod{n}$ because $\mathrm{ord}(x)$ divides $d'e' - 1$.
– 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 [Wien90], 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 [Håst86] only applies to situations where the attacker sees several messages encrypted with that exponent using different moduli. Similarly, the new class of attacks on short public exponents in [CFPR96] only applies to situations where the attacker sees the ciphertexts of several messages with a known polynomial relationship, encrypted using the same modulus.

Theorem 5. *Under the strong RSA assumption, a^{RSA} is collision-free.*

Proof sketch. An adversary who finds a collision in a^{RSA} for given n, x, i.e., who finds $y_1, \ldots, y_N$, y', and $accu'$ with

$$accu'^{y'} \equiv x^{y_1 \cdots y_N} \pmod{n},$$

can break the strong RSA assumption as follows: Let $e := y'$ and $r := y_1 \cdots y_N$. Now the e-th root y of x can be constructed as in [Sham83, BeMa94]: Compute $a, b \in \mathbb{Z}$ with $ar + by' = 1$ with the extended Euclidean algorithm (this is possible because y' is prime) and let $y := accu'^a x^b$. Thus

$$y^e \equiv accu'^{ay'} x^{by'} \equiv x^{ra+by'} \equiv x \pmod{n}.$$

□

RSA Accumulator With Random Oracle. The second example uses, as the name of the first suggests, a random oracle Ω [BeRo93]. Whenever asked to compute $\Omega(y)$ for a new value y, the oracle generates a random number r as its answer, and it stores all previous pairs (y, r) so that it answers with the same r if asked the same y again.

In practice, one replaces the random oracle by an efficient hash function. Of course, this replacement is only a heuristic.

By using a random oracle, we can construct an accumulator that is collision-free under the normal RSA assumption. The elementary accumulator uses the function

$$h^{\mathrm{RSA}\Omega}_{(n,\Omega,l)}\big(x, (y, dist)\big) := x^{2^l \Omega(y) + dist} \bmod n.$$

We do not use $\Omega(y)$ directly, because in the proof we will need that the exponents are prime numbers. So we append l bits such that $2^l\Omega(y) + dist$ is prime. Of course, this might not be possible for all values of y, so we accept only those y's as input for which a suitable $dist$ exists.

Definition 6. Let a family M of sets M_k be given where membership is decidable in polynomial time. It contains the values that we really want to accumulate for each security parameter k. The following family is called *RSA accumulator with random oracle* (for M):

- $accu\_keys^{\mathrm{RSA}\,\Omega}(k, N) := \big\{(n, \Omega, l, x) \mid n \in RSA\_Mod(k+l)$
  $\wedge\ \Omega \in \{f \mid f\colon M \to \mathbb{Z}_{n \operatorname{div} 2^l}\} \wedge l = \lceil \log_2 2k \rceil \wedge x \in \mathbb{Z}_n\big\}$
- $Y^{\mathrm{RSA}\,\Omega}_{(n,\Omega,l,x)} = \big\{(y, dist) \mid y \in M_k \wedge dist \in \mathbb{Z}_{2^l} \wedge 2^l\Omega(y) + dist \text{ prime}\big\}$, i.e., the values that we actually accumulate are pairs of a value that we want to accumulate and a suffix that turns its hash value into a prime number.
- $a^{\mathrm{RSA}\,\Omega}_{(n,\Omega,l,x)}\big((y_1, dist_1), \ldots, (y_N, dist_N)\big) :=$
  $x^{(2^l\Omega(y_1)+dist_1)\cdots(2^l\Omega(y_N)+dist_N)} \bmod n$
- $auth^{\mathrm{RSA}\,\Omega}_{(n,\Omega,l,x)}\Big((y_i, dist_i), \big(x, (y_1, dist_1), \ldots, (y_N, dist_N)\big)\Big) :=$
  $x^{(2^l\Omega(y_1)+dist_1)\cdots(2^l\Omega(y_{i-1})+dist_{i-1})\cdot(2^l\Omega(y_{i+1})+dist_{i+1})\cdots(2^l\Omega(y_N)+dist_N)} \bmod n$
- $authentic^{\mathrm{RSA}\,\Omega}_{(n,\Omega,l,x)}\big(z, (y, dist), accu\big) := ok$ iff $accu^{2^l\Omega(y)+dist} \equiv z (\bmod n)$

Theorem 7. *This accumulator is collision-free under the normal RSA assumption.*

Proof sketch. We have to show that for all N, all probabilistic polynomial-time algorithms $\widetilde{A}$, all $c > 0$, and all sufficiently large k,

$$
\begin{aligned}
\mathrm{P}\Big(& accu'^{2^l\Omega(y')+dist'} \equiv x^{(2^l\Omega(y_1)+dist_1)\cdots(2^l\Omega(y_N)+dist_N)} (\bmod n)\\
&\quad \wedge (y', dist') \notin \{(y_1, dist_1), \ldots, (y_N, dist_N)\}\\
&\quad \wedge (y', dist'), (y_1, dist_1), \ldots, (y_N, dist_N) \in \big\{(y, dist) \mid y \in M_k\\
&\qquad\qquad \wedge dist \in \{0, \ldots, 2^l - 1\} \wedge 2^l\Omega(y) + dist \text{ prime}\big\}\\
&\quad \wedge accu' \in \mathbb{Z}_n ::\\
& l := \lceil \log_2 2k \rceil; n \in_R RSA\_Mod(k+l);\\
& \Omega \in_R \{f \mid f\colon M_k \to \mathbb{Z}_{n \operatorname{div} 2^l}\}; x \in_R \mathbb{Z}_n;\\
& \big(accu', (y', dist'), (y_1, dist_1), \ldots, (y_N, dist_N)\big) \leftarrow \widetilde{A}^{\Omega}(k, N, n, l, x)\Big)\\
\le\ & \frac{1}{k^c},
\end{aligned}
$$

where $\widetilde{A}^{\Omega}$ means $\widetilde{A}$ with access to the oracle Ω. Assume that an algorithm $\widetilde{A}$ contradicts this inequality for some N. We can then construct an algorithm $\widetilde{A}_1^{\Omega}$ that calls $\widetilde{A}^{\Omega}$ and, whenever that is successful, sets

$$
\begin{aligned}
r' &:= 2^l\Omega(y') + dist' \text{ and}\\
r_i &:= 2^l\Omega(y_i) + dist_i \text{ for } i = 1, \ldots, N,
\end{aligned}
$$

and computes the r'-th root of x using the extended Euclidean algorithm for these values as in the proof of the previous theorem. The only exception is if r' equals one of the r_i's. Then an oracle collision has been found, which can only happen with very small probability. Hence it is sufficient to prove for all probabilistic polynomial-time algorithms $\widetilde{A}_1$, all $c > 0$, and all sufficiently large k,

$$\mathrm{P}\Big(y^{r'} = x \wedge r' \text{ prime} \wedge r' < n \wedge \mathit{dist}' < 2^l :: \\ l := \lceil \log_2 2k \rceil; n \in_R \mathit{RSA\_Mod}(k+l); \\ \Omega \in_R \{f \mid f\colon M_k \to \mathbb{Z}_{n \operatorname{div} 2^l}\}; x \in_R \mathbb{Z}_n; \\ (y, y', \mathit{dist}') \leftarrow \widetilde{A}_1^{\Omega}(k, N, n, l, x); r' := 2^l \Omega(y') + \mathit{dist}'\Big) \leq \frac{1}{k^c}.$$

Without loss of generality, we can assume that $\widetilde{A}_1$ has asked the oracle for $\Omega(y')$. The number of values that $\widetilde{A}_1$ asks for is bounded by a polynomial $Q(k)$. Whatever strategy $\widetilde{A}_1$ uses in choosing its oracle queries, 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 $2^l\rho + \mathit{dist}$. Thus this new adversary $\widetilde{A}_2$ is given a list of $Q(k) \cdot 2^l$ 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 to extract an e-th root for one given random e. For this, a new adversary $\widetilde{A}_3$, given e, inserts $(e \operatorname{div} 2^l)$ at a random place into a list of $Q(k) - 1$ random numbers and appends the values dist. $\widetilde{A}_3$ calls $\widetilde{A}_2$, and with a probability smaller by the factor $Q(k) \cdot 2^l$ it gets the e-th root of x (recall that $2^l \approx k$). □

The proof also shows another result that is interesting in practice, where the function used instead of the oracle is not perfect: To find an accumulator collision, one at least either has to either find a collision of this function (where collision-freeness is a much weaker requirement than "being like an oracle") or to break the strong RSA assumption.

3 Conversion Algorithm

We want to use collision-free accumulators as defined in the previous section to accumulate the public one-time keys in an FSS scheme. But what if the public one-time keys are not suitable as input for the accumulator? For example, the RSA accumulator without random oracle as defined in Section 2.3 needs prime numbers as input, and none of the known FSS schemes uses prime numbers as public one-time keys. Hence one has to convert the public one-time keys to prime numbers that can then be accumulated by the accumulator.

Of course, such a conversion could be done within the underlying one-time FSS scheme or within the accumulator. But then one has to prove their security again. Thus it seems better to use a simple conversion algorithm that has no effect on the security as an interface between the FSS scheme and the accumulator. In this way, we get a general modular construction for which one can use any collision-free accumulator and any one-time FSS scheme provided that one finds a conversion algorithm for them. As examples, we present two instantiations in Section 5. For this purpose, we use a family Λ of conversion algorithms, which has the following components:

- A function *calc_pars* that computes the security parameters k' for the accumulator and (k^*, σ^*) for the underlying FSS scheme if given as input (k, σ, N), the security parameters of the desired accumulator FSS scheme and the number of messages to be signed. The output must fulfill

$$k', k^* \geq k \text{ and } \sigma^* \geq \sigma.$$

- A polynomial-time algorithm Λ_gen which, on input k^*, σ^* and an accumulator key n, computes a key *par* specifying an individual member of Λ.
- A probabilistic polynomial-time algorithm Λ_eval which, on input a conversion key *par* and a public one-time key pk_i, outputs either a value $\widehat{pk}_i \in Y_n$ (a suitable input for the accumulator with the key n) or *"unsuitable"*. The success probability should at least be the inverse of some polynomial; in the examples, it will be at least constant.
  We write $\Lambda_{par}(pk_i)$ instead of $\Lambda\_eval(par, pk_i)$.
- A polynomial-time inversion algorithm, abbreviated Λ_{par}^{-1}, with $\Lambda_{par}^{-1}(\Lambda_{par}(pk_i)) = pk_i$ for all $\Lambda_{par}(pk_i) \neq$ *"unsuitable"*.

Note that the conversion of a one-time key is not necessarily deterministic, but the inversion has to be. So it is possible to include some random bits in the output of Λ_{par} that are needed for an accumulator, but the result of Λ_{par}^{-1} is always unique.

We now show the core of a simple example Λ_{prim}, which we will use in Section 5. It converts input numbers into prime numbers, if possible, using the same idea as in Section 2.3: The parameter *par* is a small integer l. On input $x \in \mathbb{N}$, the algorithm $\Lambda_{\mathrm{prim},l}$ checks for $dist = 1, 3, \ldots, 2^l - 1$ whether the number $2^l x + dist$ is prime. If so, it returns $2^l x + dist$, otherwise *"unsuitable"*. To get x back from the output $\hat{x}$, the inversion algorithm simply cuts off the l least significant bits.

Another example of a conversion algorithm is of course the identity function, which can be used whenever no conversion is necessary.

4 Accumulator FSS Scheme

In this section, we describe the accumulator FSS scheme. It is based on

- a one-time FSS scheme with prekey and parameters (k^*, σ^*),
- a family of collision-free accumulators with parameters (k', N), and
- a family of conversion algorithms for the one-time FSS scheme and the accumulator.

4.1 One-time FSS Scheme with Prekey

We use so-called one-time FSS schemes with prekey, e.g., [PePf97]. This prekey is generated by a center trusted by all recipients and verified by the signer, who need not trust the center. The center is used instead of the recipients themselves for simplicity. Based on this prekey, the signer can generate as many one-time key pairs as she wants. Among the two security parameters, σ^* is chosen by the signer for her information-theoretical security, whereas k^* is chosen by the center for the computational security of the recipients.

For simplicity, we only consider schemes that fulfil the simplified security criteria for schemes with prekey from [Pfit96, Theorem 7.34]. First, this means that proofs of forgery only depend on the prekey. This is natural because only the prekey is *not* chosen by the signer, i.e., a proof of forgery has to show a secret hidden in the prekey. Secondly, it is required that for every good prekey (one that the signer accepts with significant probability), for *every* one-time key pair based on it and every forgery, the probability that the forgery cannot be proved is at most $2^{-\sigma^*}$.

4.2 Construction

Key generation. The accumulator FSS scheme gets only (k, σ, N) as input. The remaining security parameters are calculated with

$$(k', k^*, \sigma^*) := \mathit{calc\_pars}(k, \sigma, N).$$

The center generates

- a prekey, using the algorithm $\mathit{gen}(k^*, \sigma^*)$ of the one-time FSS scheme.
- an accumulator key n with $n \leftarrow \mathit{accu\_gen}(k', N)$.
- the parameter for the conversion algorithm as $\mathit{par} := \Lambda\_\mathit{gen}(k^*, \sigma^*, n)$.

The signer verifies the prekey. She need not verify the accumulator key because it has no effect on her security. A weak accumulator key may make it easier for an adversary to find an accumulator collision and forge a signature. But this is no problem for the signer because she can show the collision as a proof of forgery. All these global values are part of the signer's public key, but for readability we omit them in the following.

The signer now chooses N key pairs (sk_i, pk_i) of the underlying one-time FSS scheme, based on the given prekey.

She computes $\widehat{pk}_i := \Lambda_{par}(pk_i)$ for $i = 1, \ldots, N$. If there is any $\widehat{pk}_i \notin Y_n$, i.e., $\widehat{pk}_i =$ *"unsuitable"*, she has to generate a new key pair (sk_i, pk_i) and to repeat the computation of $\widehat{pk}_i$.

Finally, the signer computes the main public key pk of the accumulator FSS scheme by accumulating the $\widehat{pk}_i$'s:

$$(pk, aux) \leftarrow a_n(\widehat{pk}_1, \ldots, \widehat{pk}_N).$$

She publishes pk and stores aux for later use. Formally, her secret key sk contains not only the secret one-time keys $sk_1, \ldots, sk_N$, but also the converted public one-time keys $\widehat{pk}_1, \ldots, \widehat{pk}_N$ and the auxiliary output aux.

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

$$s := (s_i, \widehat{pk}_i, accu_i),$$

where s_i is the one-time signature on this message with the one-time key sk_i, and $\widehat{pk}_i$ and $accu_i$ are needed for the authentication of the one-time public key pk_i. The value $accu_i$ is computed using

$$accu_i \leftarrow auth_n(\widehat{pk}_i, aux).$$

Since $accu_i$ is independent of the message, it can be precomputed when the computer is idle.

Testing. A value $s = (s_i, \widehat{pk}_i, accu_i)$ is an *acceptable signature* on the message m_i iff

1. s_i is an acceptable one-time signature on m_i with respect to $pk_i = \Lambda_{par}^{-1}(\widehat{pk}_i)$,
2. $\widehat{pk}_i \in Y_n$,
3. $accu_i \in Accu_n$, and
4. pk authenticates $\widehat{pk}_i$, i.e., $authentic_n(pk, \widehat{pk}_i, accu_i) = ok$.

Proving Forgeries. If $(s', \widehat{pk'}, accu')$ is an acceptable signature on a message m' not previously signed by the signer, she can generate a proof of forgery as follows:

1. If $pk' = \Lambda_{par}^{-1}(\widehat{pk'}) \in \{pk_1, \ldots, pk_N\}$, she tries to generate a proof of forgery in the one-time FSS scheme.
2. Otherwise, she shows the accumulator collision

   $$proof := \big((\widehat{pk}_1, \ldots, \widehat{pk}_N), (\widehat{pk'}, accu')\big).$$

   This proof shows that the assumption on which the accumulator is based has been broken.

Verifying Proofs of Forgery.

1. If $proof$ is said to be a proof of forgery in the one-time FSS scheme, one verifies that.
2. Otherwise $proof$ is accepted iff it fulfills the following conditions:
   (a) $\widehat{pk'} \notin \{\widehat{pk}_1, \ldots, \widehat{pk}_N\}$,
   (b) $\widehat{pk}_1, \ldots, \widehat{pk}_N, \widehat{pk'} \in Y_n$,
   (c) $accu' \in Accu_n$ and
   (d) $authentic_n\big(pk, \widehat{pk'}, accu'\big) = ok$ with $(pk, aux) \leftarrow a_n(\widehat{pk}_1, \ldots, \widehat{pk}_N)$.

4.3 Security

Theorem 8. *The accumulator FSS scheme as defined in the previous section is secure for both the signer and the recipients as defined in [PfWa90, PePf97].*

Proof sketch. For the information-theoretic security of the signer, we first show that any forgery that is not a forgery in the one-time FSS scheme, i.e., that does not fulfil the condition of Item 1 in "Proving Forgeries", is provable with probability 1: If $pk' \notin \{pk_1, \ldots, pk_N\}$, then $\widehat{pk'} \notin \{\widehat{pk}_1, \ldots, \widehat{pk}_N\}$ because the inversion Λ_{par}^{-1} is deterministic. Thus the value the signer computes in Item 2 is indeed an accumulator collision.

If the forgery *is* in the underlying one-time scheme, the signer can prove it with an error probability less than $2^{-\sigma^*}$, and thus less than $2^{-\sigma}$ (given that the prekey is good), because

- with probability 1, she finds the one-time key pair (sk_i, pk_i) whose public one-time key the forger has used,
- for *every* generated one-time key pair, the probability is at most $2^{-\sigma^*}$ that no proof of forgery can be found in the underlying FSS scheme, independent of the number of *"unsuitable"* public one-time keys generated before, and
- the forger gains no information about sk_i by the accumulation.

The recipients want to be secure that no signatures they have accepted become invalid. Thus it should not be possible that

- an adversary computes an acceptable signature that will be (correctly) proven to be forged by the signer, and that
- the signer can (incorrectly) deny a previously generated signature using a proof of forgery.

Hence it is sufficient to show that no proof of forgery can be computed. This is (computationally) true because a proof of forgery of the new scheme implies either a successful proof of forgery in the underlying one-time FSS scheme or a collision of the utilized accumulator. Since for both parts the security parameter is at least k (guaranteed by the function *calc_pars*), neither should be possible for a polynomially restricted forger. That some key pairs are thrown away during key generation does not help the adversary, because the proof is based on the prekey alone. □

5 Examples

We construct two examples of accumulator FSS schemes, using the two accumulators from Section 2.3. As the underlying one-time FSS scheme, we choose the one described in [HePe93]. It is based on the Discrete Logarithm assumption. Its public keys are pairs (a, b) of elements of the group where computing discrete logarithms is assumed to be hard; let their length in bits be the security parameter k^*. The algorithms of the accumulator FSS schemes are clear from the previous section as soon as we fix the conversion algorithms.

The first examples uses the accumulator a^{RSA}. It needs prime numbers as inputs, so we convert the one-time public keys (a, b) with Λ_{prim}, interpreting (a, b) as one $2k^*$-bit number.

The security parameters for the one-time FSS scheme and the accumulator are calculated by

$$(k', k^*, \sigma^*) = \mathit{calc\_pars}(k, \sigma, N) := (2k + \lceil \log_2 2k \rceil + 1, k, \sigma),$$

and the key of the conversion algorithm by

$$l = \Lambda\_gen(k^*, \sigma^*, n) := \lceil \log_2 2k^* \rceil.$$

These functions guarantee that the converted public one-time keys are in the domain of the accumulator: The parameters for the one-time FSS scheme are simply the given k and σ. The parameter k' for the accumulator is set such that the RSA modulus is longer than a one-time FSS key and the appended value *dist*. The length l of *dist* is a somewhat arbitrary value ensuring that a prime will typically be found in the search interval.

The second example is based on the RSA accumulator with a random oracle assumption. This accumulator needs pairs $(pk_i, dist_i)$ as input, so the conversion algorithm is similar to $\Lambda_{\mathrm{prim},l}$, but returns $(pk_i, dist_i)$ instead of $2^l \Omega(pk_i) + dist_i$ if that value is prime. The security parameters are computed with

$$(k', k^*, \sigma^*) = \mathit{calc\_pars}(k, \sigma, N) := (k, k, \sigma)$$

and the key of the conversion algorithm is

$$\Lambda\_gen\big(k^*, \sigma^*, (n, \Omega, l, x)\big) := (l, \Omega).$$

Concretely, this means that the length of the RSA modulus used for the accumulator is independent of the length of the one-time keys, because only oracle outputs with appended values *dist* are accumulated, and the length of the oracle output is adapted accordingly.

6 Conclusion

We have presented a generalized definition of accumulators and the definition of a collision-free subtype. We constructed two collision-free accumulators, one based on a stronger RSA assumption than usual, the other based on a random oracle and the normal RSA assumption. We remind the reader that no new assumption in cryptology should be trusted, i.e., we certainly do not recommend the first version for use in practice for quite some time. These accumulators can be used to construct fail-stop signature schemes in which the length of the public key and of the signatures is independent of the number N of messages that can be signed, while the additional cost for signing is small, especially because most of the signature can be computed and sent before the message is known.

Key generation, however, takes significantly longer than in constructions with trees. To avoid the precomputation of a very long secret key, one can combine the constructions with top-down tree authentication. In this way, we get flat trees similar to those in [DwNa94]. For instance, one might use accumulation for 1024 pairs (sk_i, pk_i) each, form a tree with two levels of such structures, and generate the structures of the lower level on demand, signing their "public" keys with the secret keys of the upper level. Thus one can sign one million messages with one public key. A complete signature consists of the accumulation result z of one lower-level structure and two accumulator FSS signatures as described in Section 4.

Acknowledgments

We thank Michael Waidner, Joachim Biskup, Andreas Pfitzmann, and Ute von Jan for helpful comments on this paper.

References

[Bari96] Niko Barić: *Digitale Signaturen mit Fail-stop Sicherheit ohne Baumauthentifizierung.* Diplomarbeit, Institut für Informatik, Universität Hildesheim, July 1996.

[BeMa94] Josh Benaloh and Michael de Mare: *One-Way Accumulators: A Decentralized Alternative to Digital Signatures.* In *Advances in Cryptology — EUROCRYPT '93*, LNCS 765, pages 274–285. Springer-Verlag, Berlin, 1994.

[BeRo93] Mihir Bellare and Phillip Rogaway: *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols.* In *1st ACM Conference on Computer and Communications Security, November 1993*, pages 62–73. acm press, New York, 1993.

[CFPR96] Don Coppersmith, Matthew Franklin, Jacques Patarin, and Michael Reiter: *Low-Exponent RSA with Related Messages.* In *Advances in Cryptology — CRYPTO '96*, LNCS 1070, pages 1–9. Springer-Verlag, Berlin, 1996.

[CrDa96] Ronald Cramer and Ivan B. Damgård: *New Generation of Secure and Practical RSA-Based Signatures.* In *Advances in Cryptology — CRYPTO '96*, LNCS 1109. Springer-Verlag, Berlin, 1996.

[DwNa94] Cynthia Dwork and Moni Naor: *An Efficient Existentially Unforgeable Signature Scheme and its Application.* In *Advances in Cryptology — CRYPTO '94*, LNCS 839, pages 234–246. Springer-Verlag, Berlin, 1994.

[Håst86] Johan Håstad: *On Using RSA with Low Exponent in a Public Network.* In *Advances in Cryptology — CRYPTO '85*, LNCS 218, pages 403–408. Springer-Verlag, Berlin, 1986.

[HePe93] Eugène van Heyst and Torben P. Pedersen: *How to Make Efficient Fail-stop Signatures.* In *Advances in Cryptology — EUROCRYPT '92*, LNCS 658, pages 366–377. Springer-Verlag, Berlin, 1993.

[HePP93] Eugène van Heijst, Torben P. Pedersen, and Birgit Pfitzmann: *New Constructions of Fail-Stop Signatures and Lower Bounds.* In *Advances in Cryptology — CRYPTO '92*, LNCS 740, pages 15–30. Springer-Verlag, Berlin, 1993.

[Nybe96a] Kaisa Nyberg: *Commutativity in Cryptography.* In *Proceedings of the First International Workshop on Functional Analysis at Trier University*, pages 331–342. Walter de Gruyter, Berlin, 1996.

[Nybe96b] Kaisa Nyberg: *Fast Accumulated Hashing.* In *3rd Fast Software Encryption Workshop*, LNCS 1039, pages 83–87. Springer-Verlag, Berlin, 1996.

[PePf97] Torben P. Pedersen and Birgit Pfitzmann: *Fail-Stop Signatures.* to appear in SIAM Journal on Computing, 26(2):291–330, April 1997.

[Pfit94] Birgit Pfitzmann: *Fail-Stop Signatures Without Trees.* Hildesheimer Informatik-Berichte 16/94, ISSN 0941-3014, Institut für Informatik, Universität Hildesheim, June 1994.

[Pfit96] Birgit Pfitzmann: *Digital Signature Schemes — General Framework and Fail-Stop Signatures.* LNCS 1100. Springer-Verlag, Berlin, 1996.

[PfWa90] Birgit Pfitzmann and Michael Waidner: *Formal Aspects of Fail-stop Signatures.* Interner Bericht 22/90, Fakultät für Informatik, Universität Karlsruhe, December 1990.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard Adleman: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120–126, February 1978.

[Sham83] Adi Shamir: *On the Generation of Cryptographically Strong Pseudorandom Sequences.* ACM Transaction on Computer Systems, 1(1):38–44, February 1983.

[WaPf90] Michael Waidner and Birgit Pfitzmann: *The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability.* In *Advances in Cryptology — EUROCRYPT '89*, LNCS 434, page 690. Springer-Verlag, Berlin, 1990.

[Wien90] Michael J. Wiener: *Cryptanalysis of Short RSA Secret Exponents.* IEEE Transactions on Information Theory, 36(3):553–558, May 1990.