New Generation of Secure and Practical RSA-Based Signatures

Ronald Cramer*
Ivan Damgård**

Abstract. For most digital signature schemes used in practice, such as ISO9796/RSA or DSA, it has only been shown that certain plausible cryptographic assumptions, such as the difficulty of factoring integers, computing discrete logarithms or the collision-intractability of certain hash-functions are necessary for the security of the scheme, while their sufficiency is, strictly speaking, an open question.
A clear advantage of such schemes over many signature schemes with security proven relative to such common cryptographic assumptions, is their efficiency: as a result of their relatively weak requirements regarding computation, bandwidth and storage, these schemes have so far beaten proven secure schemes in practice.
Our aim is to contribute to the bridging of the gap that seems to exist between the theory and practice of digital signature schemes. We present a digital signature that offers *both* proven security and practical value. More precisely, under an appropriate assumption about RSA, the scheme is proven to be not existentially forgeable under adaptively chosen message attacks. We also identify some applications where our scheme can be conveniently implemented using dedicated smartcards that are available today.

1 Introduction

Consider, very generally, electronic transaction systems that require message authentication mechanisms such as digital signature schemes. Although we do not mean to limit ourselves to this case in this paper, assume that the individual players have dedicated (i.e., capable of performing public key cryptography) smartcards as available today or in the near future, as their user-devices. We will simply say that a digital signature scheme has practical value in this context, if it can be conveniently used, given the available infrastructure *and* hardware.

Our objective is to design a digital signature scheme that offers *both* high security and practical value. Informally, we require the following of our target scheme. First, relative to some plausible cryptographic assumption, a proof must be given that the scheme is not existentially forgeable under adaptively chosen

* CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: cramer@cwi.nl.
** Aarhus University & BRICS, Ny Munkegade, Aarhus, Denmark. Email: ivan@daimi.aau.dk.

message attacks [13]. Without attempting to quantify the efficiency needed, we require, secondly, that the amount of computation and the size of the signatures are small, and, finally, that the amount of storage needed is reasonably limited.

In a sequence of results [17], [1], [18] and finally [23], it was established that the existence of one-way functions is necessary and sufficient for the existence secure signatures. This result, however theoretically very important, does not give rise to a practical signature scheme. The construction, which is based on a general *one-way function*, uses a costly "bit-by-bit" signing technique in conjunction with tree authentication [17]. As a result, the size of signatures is $O(k^2 \cdot \log i)$, where k stands for a security parameter and i indicates the number of signatures made.

Benefitting from the special properties of *claw-free trapdoor permutations*, the secure scheme presented in [13] achieves signatures of size $O(k \cdot \log i)$ instead. Their scheme also uses a tree structure. Intractability of factoring is a sufficient assumption for the existence of the family of functions required for their scheme (for an extension of their result, see [7]). Though yielding shorter signatures asymptotically, the size grows rapidly in practice as the number of signatures made increases.

Starting with the seminal paper [22], which proposed the RSA-functions as the first implementation of public-key cryptography as envisaged by Diffie and Hellman [9], many practical digital signature schemes have been proposed, for instance, [11], [12], [24], [14], [20], [16] and [19].

Although many of them are actually used in practice today, these schemes seem to have the property that their security is hard to analyze. We certainly do not mean to suggest here that their security is dubious. On the contrary, these schemes rely on common cryptographic assumptions, such as the difficulty of factoring or inverting RSA-functions, the difficulty of computing discrete logarithms or the collision intractability of certain hash functions, and have so far resisted many years of cryptanalytic efforts.

However, none of these practical schemes has been shown to be secure in the sense of [13] provided that any of these mentioned cryptographic assumptions holds. This implies that, independently of their validity, these necessary and common cryptographic assumptions may still turn out to be insufficient for the security of these signature schemes. Thus, based on the above, it is still an open problem to design a secure and truly practical digital signature scheme, that may be used in today's or tomorrow's information systems.

Recently, progress has been made in this area. Starting with [10], it can be concluded that the first two requirements, namely proven security, moderate amount of computation and provision of any reasonable number of small-sized signatures, can be satisfied. The cryptographic assumptions needed there, are an RSA-assumption and the factoring assumption (or more precisely, the existence of a particular family of claw-free trapdoor permutations), respectively. For efficient *fail-stop signatures*, see [21]. These schemes yield practically much smaller signatures compared to, for instance, [13]. The reason is that, instead of binary authentication trees, these schemes allow the use of trees with much larger branching degree.

Briefly, the efficiency of this scheme is as follows. Let integers l, d and a security parameter k be given (in [10] it must be required that $l \geq k$). In [10], a signer can make at least l^d signatures. The size of a signature in [10] amounts to dk bits. [3]. The idea is then to choose l large, such that for any reasonable number of signatures the resulting size of the signatures is reasonably small.

Theoretically, this scheme offers a trade-off, via the flexibility of choosing l, between the size of signatures and the storage required: the size is $O(\frac{k}{\log l} \cdot \log i)$ bits, with $O(l \cdot k)$ bits storage for the system constant and $O(\frac{k}{\log l} \cdot \log i)$ bits dynamic storage for the signer. The corresponding figures for [13] and [7], are $O(k \cdot \log i)$ bits for the size of signatures and $O(k \cdot \log i)$ bits storage [4]. A disadvantage that in [10] may be that all signers and receivers of signatures must have access to a large list of random numbers. This lists consists of l random k-bit strings and l primes.

In [10], authentication of computer faxes is identified as an application where their proposed scheme is certainly useful. However, in any practical system that uses smartcards as the main players, this assumption about shared access to the list of random numbers may be too demanding, simply because of its storage requirements (in case a user has a *wallet with observer* ([5], [6]) as user device, there are solutions, though not as efficient as the scheme presented in this paper, that preserve the off-line property). One can envision a system where the players gain access to the list through a server. If this server and the communication link are trusted, this solution has only the on-line character as the main disadvantage. Otherwise, one also has to employ mechanisms for ensuring the integrity of the supplied data (one-way accumulators [2] seem to allow for an efficient approach).

Our contribution is the design of a secure signature scheme where the size of the signatures is $(d+1)k$ bits, while l^d signatures can be made. The integers l and d can be chosen independently from the security parameter k. The security is derived from an appropriate RSA-assumption. Technically, our scheme builds on [10]. Our improvement over [10] resides in the fact that our scheme does not require the players to share a list of l random strings of size k bits; they only need to share a list consisting of $l+1$ primes.

As an example, take a 1000-bit RSA-modulus, and set $l = 1000$ and $d = 3$. In this case, a signer can make over a billion signatures, the size of each not exceeding 4000 bits. The secret key has size 1000 bits, while the public key is 3000 bits. Now, let the shared list consist of the first 1001 consecutive odd primes. By storing the differences between consecutive primes, this requires hardly any storage. In [10], this particular choice for the list of primes has also been proposed. But there, the players would *additionally* have to share 1 million random bits. The required assumption about RSA is the same in both cases.

This example indicates that our secure scheme may very well be implemented

[3] The actual sizes stated in [10] are larger. However, these can be reduced by roughly a factor of two if one observes that the signatures are redundant, i.e., part of the signature can be recalculated from another part. See also Section 3.

[4] The dynamic storage can be reduced by applying a suggestion from [3].

in a system that uses today's dedicated smartcards.

More generally, our scheme works with any list of primes shared between the players, but to limit the storage, it is convenient to take consecutive primes. By choosing a random sequence of $l+1$ consecutive primes, the security of our scheme is equivalent to the general RSA-assumption.

Our exposition is organized as follows. In Section 2, we outline the technical ideas behind our design. The formal presentation of the scheme can be found in Section 3. The latter section left open the choice of some parameters. This is resolved in Section 4, which is followed by a discussion of the performance of our scheme in Section 5. The proof of security is given in Section 6. In Section 7, we give optimizations of the proposed scheme that cut the storage requirements even further.

2 Basic Ideas

Conceptually, our signature scheme may be viewed as a cross between [13] and [10], together with modifications enabling their synthesis. Let l and d be integers. In [10], all players in the signature scheme must have access to two lists. The first list contains l primes. Depending on the particular RSA-assumption one wishes to make, these could be, for instance, the first l odd primes, or l random primes. The second list consists of l random k-bit strings. Here, k is a security parameter and l is an integer with $l \geq k$. Our first objective is to remove the necessity of the list of random numbers.

In [10], the system constants are as follows. Let L denote the list of primes $\{p_0, \ldots, p_{l-1}\}$, L' the list of l random l-bit strings $\{x_0, \ldots, x_{l-1}\}$ and let a denote a random l-bit string, to be used as the root of all authentication trees. Let a signer be given an RSA-modulus n together with its factorization. The public key consists of n and y_{root}. The latter is to be the root of an l-ary authentication tree of depth d. The factorization of n is private input for the signer.

The "basic authentication step" in [10] is

$$y \leftarrow (\alpha \cdot \prod_{i=0}^{l-1} x_i^{\beta_i})^{\frac{1}{p_j}} \bmod n$$

where α is an already authenticated value, $\beta = \beta_0 || \cdots || \beta_{l-1}$ is an l-bit string to be authenticated, and p_j is a prime from the list L that has not been used before in connection with α. Instead, our basic authentication step is

$$y \leftarrow (\alpha \cdot h^{\beta})^{\frac{1}{v_j}} \bmod n,$$

where h is a member of $\mathbb{Z}_n^*$ and part of the signer's public key. Furthermore, e_j is the smallest integer such that $v_j \equiv p_j^{e_j} > n$. Here the values that can be authenticated are elements of $\mathbb{Z}_n^*$. This removes the list L' and the condition that $l \geq k$. However, implementing this idea only results in a scheme that we can prove secure against random message attacks. Such a scheme can be efficiently transformed to a scheme that is secure against active attacks, as is desired here, by means of a technique described in [8]. The loss of efficiency is a factor of two (twice as much computation, signature size twice as large). But we can do better in this case, if we add one prime q with a special purpose to the list: it is only used when a message m, agreed upon between the signer and a receiver, is to be authenticated, as follows

$$z \leftarrow (\alpha \cdot h^m)^{\frac{1}{w}} \bmod n,$$

where α is a "freshly" generated leaf in the authentication tree and e is the smallest integer such that $w \equiv q^e > n$. This relates to the idea [13] of applying sufficiently independent functions to the actual signing process and the construction of an authentication tree, respectively.

To minimize the storage needed for the list of primes, we can take L to consist of l consecutive primes. Then, only the first prime and all consecutive differences are stored. In Section 7, two other techniques are given for further improvements of the effciency of the scheme.

3 Description of the Scheme

In a preprocessing-phase, a security parameter k is determined, as well as integers l and d. Next, a list L consisting of $l+1$ distinct primes is generated by invoking an algorithm $H(1^k, 1^l)$, say $L = \{q, p_0, \dots, p_{l-1}\}$. Ways of choosing H are discussed in the next section.

Furthermore, we assume that we are given a probabilistic polynomial time generator G that, on input 1^k, outputs a triple (n, r, s), where r and s are primes and $n = r \cdot s$ is a k bits integer. It is assumed that G is defined such that it is infeasible to factor n, when only n as generated by G is given as input. Finally, we must have that q and the p_i are co-prime to $\phi(n)$. Given n and L, define e as the smallest integer such that $q^e > n$ and e_i as the smallest integer such that $p_i^{e_i} > n$ for $i = 0 \dots l-1$. In the following, w denotes q^e and v_i denotes $p_i^{e_i}$, for $i = 0 \dots l-1$.

We start with an informal overview of the scheme. The signer has as public key an RSA-modulus n, $h \in \mathbb{Z}_n^*$ and $x_0 \in \mathbb{Z}_n^*$. Here, n is generated by $G(1^k)$ and h and x_0 are chosen at random from $\mathbb{Z}_n^*$ by the signer. In a possible variation of the scheme, x_0 and h are chosen mutually at random and are the same for all signers. In any case, h and x_0 must be chosen at random to avoid weak keys.

As always, his knowledge of the factorization (r, s) of n enables the signer to compute $X^{\frac{1}{u}} \bmod n$ for any $X \in \mathbb{Z}_n^*$ and any integer u such that $\gcd(u, (r-1)(s-1)) = 1$. The public key consists of the triple (n, h, x_0). The factorization of n is private input to the signer.

The algorithm DFS(i), which is used in the formal description of our scheme, gradually develops a full l-ary tree of depth d by selecting the nodes at random from $\mathbb{Z}_n^*$. The tree is constructed in depth-first fashion. Although not explicitly given as input to DFS(i), it is assumed that it has access to l, d, x_0 and n. The value x_0 serves as the root of the tree. Each time DFS(i) is invoked ($i = 1 \dots l^d$), it creates a path to a new leaf x_d and outputs this path, say, $x_1, \dots, x_d$ (the root

x_0 being understood). This sequence is ordered such that x_{j-1} is the parent of x_j $(j = 1 \ldots d)$.

Furthermore, for each node x_j in this sequence, DFS(i) also outputs an indicator i_j $(j = 1 \ldots d)$ in such a way that i_j is assigned to x_j if and only if x_j is the i_j-th child of x_{j-1}. The amount of storage needed for this procedure (apart from l, d, x_0 and n) does not exceed the amount of storage needed for $d-1$ pairs consisting of a node and an indicator.

By invoking DFS, the signer gradually constructs, in a depth first fashion, an l-ary authentication tree with depth d: each time a new signature is required he constructs a path to a new leaf. All nodes x are members of $\mathbb{Z}_n^*$, given by their smallest non-negative representative modulo n. The message space is equal to the set $\{0,1\}^k$, which we will also identify with the set of non-negative integers smaller than 2^k.

In Figure 1, the signer is making his i-th signature, on a message $m \in \mathbb{Z}_n^*$. So, in particular x_d is the i-th leaf he reaches. The part of the tree on the right side of the path $x_0, \ldots, x_{d-1}, x_d$ is not yet constructed. Since x_1 happens to be the i_1-st child of x_0, the signer authenticates x_1 with respect to the prime p_{i_1} by computing $y_1 \leftarrow (x_0 \cdot h^{x_1})^{\frac{1}{v_{i_1}}} \bmod n$. Similar rules apply to the authentication of the remaining nodes in this path. In particular, it so happens to be in our example that x_d is the i_d-th child of x_{d-1}. Thus x_d is authenticated by computing $y_d \leftarrow (x_{d-1} \cdot h^{x_d})^{\frac{1}{v_{i_d}}} \bmod n$. Finally, the message m is signed by computing $z \leftarrow (x_d \cdot h^m)^{\frac{1}{w}} \bmod n$. Notice that the prime q is only used when the "actual signature" is computed, while the other primes in the list L are used exclusively in the process of constructing the authentication tree. The signature on m consists of the y_j and indicators i_j, $(j = 1 \ldots d)$ and z.

Concerning the storage needed for the signer, notice that the part of the tree left from the path $(x_0, \ldots, x_{d-1}, x_d)$ can be deleted. Actually, x_d itself can be removed. In order to carry on with the depth-first construction of the tree, it is sufficient to store $x_0, \ldots, x_{d-1}$ and the indicators to their parents. This storage amounts to at most $(d-1)(k + \log l)$ bits (the root x_0 is part of the public key).

A receiver of this signature gets only the message m, authentication values y_j, the indicators i_j $(j = 1 \ldots d)$ and z. So, what about the nodes? These are re-computed as follows. On input of the public key, the list L, m and z he recomputes x_d as $x_d \leftarrow z^w \cdot h^{-m} \bmod n$. Recursively, the receiver re-computes x_{j-1} from x_j, y_j and i_j in a similar fashion $(j = d \ldots 1)$. The last node x_0 he thus computes should be equal to the actual x_0, which is part of the public key. If so, the signature is accepted. We point out that in many tree-structured signature schemes, it is sufficient to send the authentication values and have the verifier re-compute the nodes, instead of defining these as part of the signature. It is easily seen why this does not affect the security at all: briefly, if the verifications in the "reduced" scheme hold, one gets a signature in the original scheme (on the same message, of course) by simply incorporating the re-computed nodes. The remark in a footnote in Section 1 is based on this observation.

More formally, the description of the new signature scheme is as follows.

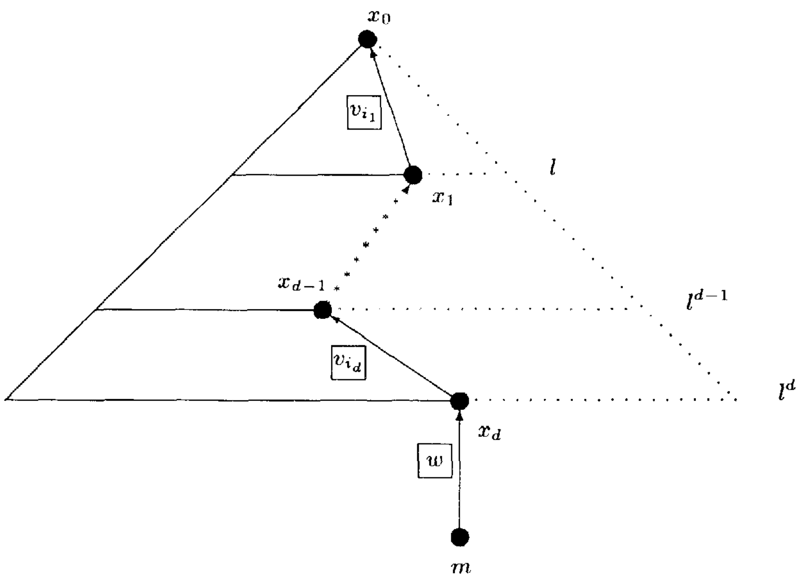

Fig. 1. The i-th Signature

Preprocessing:
A security parameter k, integers l and d are determined. Next, the system constant $L = \{q, p_0, \ldots, p_{l-1}\}$ consisting of $l+1$ distinct primes is generated by invoking $H(1^k, 1^l)$. Define e as the smallest integer such that $w \equiv q^e > n$, and e_i as the smallest integer such that $v_i \equiv p_i^{e_i} > n$, for $i = 0 \ldots l-1$. For possible choices of H, see Sections 4 and 7.

Initialization:
The signer runs $G(1^k)$ and obtains a triple (n, r, s) such that q and the p_i are co-prime to $\phi(n)$. Next, he chooses h and x_0 at random in $\mathbb{Z}_n^*$. His public key pk is now the pair (n, h, x_0), while his secret key sk consists of the pair (r, s).

Signing:
Let a k bit message m be given. Then the i-th signature, where $1 \le i \le l^d$, is computed as follows. First, the signer puts $(x_1, i_1, \ldots, x_d, i_d) \leftarrow \mathrm{DFS}(i)$. Next, he computes (for $j = 1 \ldots d$) $y_j \leftarrow (x_{j-1} \cdot h^{x_j})^{\frac{1}{v_{i_j}}} \bmod n$. Finally, he computes $z \leftarrow (x_d \cdot h^m)^{\frac{1}{w}} \bmod n$. The signature σ on m consists of the values $z, y_1, i_1, \ldots, y_d, i_d$.

Verification: Verification is done as follows. The receiver of a signature puts $\sigma = (Z, Y_1, i_1 \ldots, Y_d, i_d)$, and, on input of $pk = (n, h, x_0)$, m and σ, he computes $X_d \leftarrow Z^w \cdot h^{-m} \bmod n$. Finally, he computes $X_{j-1} \leftarrow Y_j^{v_{i_j}} \cdot h^{-X_j} \bmod n$ $(j = d \ldots 1)$. If $X_0 \equiv x_0 \bmod n$, the signature is accepted.

Remark 1 *For convenient exposition of the scheme, we have chosen to let the signer only use the leaves for signing. However, the scheme is easily adapted so* *as to allow for a more extensive use of the authentication tree. In this modified scheme, each freshly constructed node can immediately be used for making a signature. The proof of security is easily adapted to fit with this modification.*

4 Generating the List of Primes L

In order to minimize the storage needed for the system constants, i.e., the list L, k, l and d, it is convenient to set L to any $l+1$ consecutive primes greater than 2. In this case, only the first prime, the differences between consecutive primes and the exponents e and e_i are stored. As an example, one could take L to consist of the first $l+1$ (odd) primes.

It must be stressed, however, that the correctness of the scheme is independent of the particular ways of generating L. Also, the proof of security is not affected by such choices. What is dependent on the choice of L, is the particular assumption we have to make about RSA-inversion. See Section 6.

5 Performance of the Scheme

A signer can make at least l^d signatures (see also Remark 1) such that the size of each signature does not exceed $(d+1)k$ bits (neglecting the $d \log l$ bits needed to indicate the path). A public key has size $3k$ bits.

Concerning the amount of computation needed, signing requires two full RSA-exponentiations and one modular multiplication on-line. A path to the current leaf can be authenticated by pre-processing, using $2d$ full-RSA exponentiations and d modular multiplications. A receiver of a signature will have to perform $2(d+1)$ full RSA-exponentiations and $d+1$ modular multiplications.

For the gradual depth-first construction of the authentication tree, the signer stores at most $(d-1)(k+\log l)$ bits at any time. Secure storage in the strongest sense (i.e., storage not accessible or alterable by "the outside world") is only needed for the secret key (k bits) and the relevant nodes of the latest path in the tree, which is at most $(d-1)k$ bits. The public list L only has to be securely stored in a weaker sense: the signer must have certainty that L is authentic.

6 Proof of Security

The proof of security works for any choice of the list L. However, the particular assumption we make about the difficulty of RSA-inversion depends on this very choice in the following way. We require the following of the algorithm H.

Assumption 1 *Let k be a security parameter and let l be of polynomial size in k. Let L be generated by $H(1^k, 1^l)$ and let n be an RSA-modulus as generated by $G(1^k)$ and let x be a random member of $\mathbb{Z}_n^*$. Then there is no probabilistic polynomial time algorithm that has non-negligible probability of computing $x^{\frac{1}{\alpha}} \bmod n$ with $\alpha \in L$, on input L, n and x.*

Under this assumption, we can prove that the signature scheme is not existentially forgeable under adaptively chosen message attacks.

Theorem 1 *Under Assumption 1, the signature scheme presented in Section 3 is not existentially forgeable under adaptively chosen message attacks.*

Proof: We are given integers l and d, a list $L = \{q, p_0, \ldots, p_{l-1}\}$ consisting of $l+1$ distinct primes and an RSA-modulus n. Let w and v_i be defined as in Section 3, for $i = 0 \ldots l-1$. We assume that n is generated according to $G(1^k)$, but we are not given the factorization. Also, we assume that q and the p_i are co-prime to $\phi(n)$ and that L is generated according to $H(1^k, 1^l)$. The proof is by contradiction. We show that existence of a successful attacker implies that we can compute $X^{\frac{1}{\alpha}}$ mod n, given a random $\alpha \in L$, and a random $X \in \mathbb{Z}_n^*$. Which contradicts Assumption 1.

Let $\alpha \in L$. First, we show that we can set up a "simulated" signer, who as input $h \in \mathbb{Z}_n^*$ and $h^{\frac{1}{\beta}}$ mod n for all β in L different from α, but is yet indistinguishable from a signer who proceeds as in Section 3 after he is given h, n *and* its factorization. To this end, we consider two cases separately and focus mainly on the differences with Section 3. Finally, we run the attacker against this simulated signer and obtain the desired contradiction.

Technically, the simulation runs as follows. In case $\alpha = w$, the root x_0 is computed as $x_0 \leftarrow a_0^{v_0 \cdots v_{l-1}}$ mod n, for randomly chosen a_0 from $\mathbb{Z}_n^*$. The value a_0 is stored. All nodes x, excluding the leaves, are computed as $x \leftarrow a^{v_0 \cdots v_{l-1}}$ mod n, where a is chosen at random from $\mathbb{Z}_n^*$. The value a is stored. If any x is the f-th child of his parent $x_* = a_*^{v_0 \cdots v_{l-1}}$ mod n, the authentication value y is computed as $y \leftarrow a_*^{v_0 \cdots v_{f-1} \cdots v_{f+1} \cdots v_{l-1}} \cdot (h^{\frac{1}{v_f}})^x$ mod n. After the i-th signature on a message m, the i-th leaf x is computed as $x \leftarrow a^w \cdot h^{-m}$ mod n where a is chosen at random from $\mathbb{Z}_n^*$. Next, the simulated signer reveals the path to the i-th leaf, together with all authentication values, and the authentication value $z = a$ of the message m.

In case $\alpha \neq w$, say, $\alpha = v_j$, the authentication tree has to be constructed from the bottom up. We first show how this is done for $d = 1$. We select the j-th child at x at random from $\mathbb{Z}_n^*$. The parent x_* is then computed as $x_* \leftarrow b^{v_0 \cdots v_{l-1}} h^{-x}$ mod n, where b is chosen at random from $\mathbb{Z}_n^*$. The value b is stored. The authentication value y of x is computed as $y \leftarrow b^{v_0 \cdots v_{j-1} v_{j+1} \cdots v_{l-1}}$ mod n. Finally, the remaining $l-1$ children of x_* are selected at random from $\mathbb{Z}_n^*$. Let x' be the f-th child ($f \neq j$). Then its authentication value y' is computed as $y' \leftarrow b^{v_0 \cdots v_{f-1} v_{f+1} \cdots v_{l-1}} \cdot (h^{\frac{1}{v_f}})^{x'-x}$ mod n. When we have constructed $l-1$ other such trees with $d = 1$, the same procedure can be used to combine them into a tree with $d = 2$, by letting the roots play the role of the leaves as above. By induction, we can build an l-ary tree with any depth d.

One choice has been left open in the present case. The leaves x of the target tree of depth d must be chosen as $x \leftarrow b^w$ mod n, for random b in $\mathbb{Z}_n^*$. With the i-th signature request, the simulated signer can reveal the path to the i-th leaf, together with all authentication values, and the authentication value $z \leftarrow b \cdot (h^{\frac{1}{w}})^m$ mod n.

It is clear that in both cases each node in the tree has the uniform distribution and is independent of anything else. All other values follow deterministically. Thus this simulation cannot be distinguished from the real signer.
In the next step in our proof, we run the attacker against the simulated signer and show that we can compute $X^{\frac{1}{\alpha}} \bmod n$, for random $\alpha \in L$, and a random $X \in \mathbb{Z}_n^*$. Here, we have essentially the same success-probability as the attacker. Recall that n and L were generated by G and H respectively.
We proceed as follows. We choose a random $\alpha \in L$, a random $X \in \mathbb{Z}_n^*$ and a random ρ from $\mathbb{Z}_n^*$. Put $h \leftarrow X^{\prod_{\beta \in L/\{\alpha\}} \beta} \cdot \rho^{\prod_{\beta \in L} \beta} \bmod n$. Next we feed L, n, h, and $h^{\frac{1}{\beta}} \bmod n$ for all β in L different from α to the simulated signer and run the simulation (note that h is also distributed as in "real life") Next, we run the attacker against this simulator. Assume that after l^d calls to the simulated signer, the attacker outputs a forgery [5]

$$\tilde{m}, \tilde{z}, x_0, \tilde{x}_1, i_1, \tilde{y}_1, \ldots, \tilde{x}_d, i_d, \tilde{y}_d,$$

i.e., a signature on a message $\tilde{m}$ that has not been signed by the simulator in the course of the attack. Now, let T denote the full-tree of depth d and branching l that the simulated signer has output in the course of the attack. Define j to be the largest integer such that $x_0, \tilde{x}_1, i_1, \ldots, \tilde{x}_j, i_j$ is a path in T. If $j = d$, then $\tilde{x}_d$ is a leaf. So, there exists a signature

$$m, z, x_1, i_1, y_1, \ldots, x_d, i_d, y_d,$$

output by the simulated signer, such that $\tilde{x_d} = x_d$. By the assumption on $\tilde{m}$, we have $\tilde{m} \neq m$. So, we have

$$(y_d \cdot \tilde{y}_d^{-1})^q \equiv h^{m-\tilde{m}} \bmod n.$$

But since $m - \tilde{m} \neq 0 \bmod w$ (recall that we have $0 \leq m, \tilde{m} < n$, while w and the v_i are greater than n), we can easily extract $h^{\frac{1}{w}} \bmod n$ from this as follows. Put $m - \tilde{m} \bmod w = q^j \cdot e$, with $\gcd(q, e) = 1$ and $0 \leq j \leq e-1$. Let the integers f and i be such that $e \cdot f = 1 + i \cdot q^{e-j}$. Then $h^{\frac{1}{q}} = (y_d^f \cdot \tilde{y}_d^{-f} \cdot h^i)^{q^{e-j-1}}$.
If, on the other hand, $j < d$, then $\tilde{x}_j$ is a node in T at depth j and $\tilde{x}_{j+1}$ is not a child of $\tilde{x}_j$ in T. Let x_{j+1} denote the i_{j+1}-th child of $\tilde{x}_j$ in T. Then clearly, by assumption on j, $x_{j+1} \neq \tilde{x}_{j+1}$. Thus,

$$(y_{j+1} \cdot \tilde{y}_{j+1}^{-1})^{v_{i_{j+1}}} \equiv h^{x_{j+1} - \tilde{x}_{j+1}} \bmod n,$$

with $x_{j+1} - \tilde{x}_{j+1} \neq 0 \bmod v_{i_{j+1}}$. From this value, $h^{\frac{1}{p_{i_{j+1}}}} \bmod n$ is extracted as above in the case $j = d$. We conclude that the forgery allows us to compute $h^{\frac{1}{\alpha}} \bmod n$ for some $\alpha \in L$. By the construction of h, it follows, by the same calculations as above, that we can efficiently derive $X^{\frac{1}{\alpha}} \bmod n$ from this value.

[5] In the verification, the receiver of the signature checks if the signature consists of d nodes. We can remove this "length-check"-condition at the expense of a slightly more technical proof than presented here.

From the perfectness of the simulation the probability that $\alpha = \beta$ is $\frac{1}{l+1}$. Thus, if the attacker has non-negligible success- probability, then we can extract random α-th roots also with non-negligible probability, for $\alpha \in L$. □

Note that if a signer deviating from the signer's algorithm, should deliberately compute two messages that have the same signature, a receiver can easily compute a multiple of the order of h, which may allow that receiver to forge or even factor the signer's modulus.

In the Section 4 we have suggested to make a particular choice that minimizes the storage of L, namely of having L consist of $l+1$ consecutive primes. Furthermore, for reasons of simplicity, we have suggested that these are the first $l+1$ primes of size $k+1$ bits.

7 Optimizations

In this section, we describe a number of provably secure methods for decreasing the required size of the exponents in the list L (See also Section 2).

Using Multiple Values of h

In this variation, the signer generates two values h as described in Section 3, h_1 and h_2. Let β be some k bits string that has to be authenticated. The signer splits β into two blocks β_1 and β_2 of size $\frac{k}{2}$ bits each and computes the authentication value for β as follows.

$$y \leftarrow (\alpha \cdot h_1^{\beta_1} h_2^{\beta_2})^{\frac{1}{p}} \bmod n,$$

for some appropriate exponent p and node α. This cuts the required size of the exponents by a factor of two. The expenses are an increase of the size of the public key by k bits. As noted before, the value of h may be chosen mutually at random between the signers. This also holds for this method, and as such it would mean an increase of k bits of the system constant. This method preserves the security properties of the scheme, and can be used in conjunction with the other methods presented.

Using a Hash-Function

Let $\mathcal{H}$ be a collision-resistant hash-function that maps arbitrary sized input strings to strings of size $k_* << k$. All values to be authenticated in the signature scheme, i.e., the nodes in the tree and the messages, are to be hashed down to k_* bits first. Also, the root of the authentication tree as part of the public key, can be replaced by a hash of that root.

The effect is that the required size of the exponents is now k_* bits instead of k bits. The security statement now also requires that $\mathcal{H}$ is collision-resistant. This method can be used in conjunction with any of the other methods presented in this paper.

References

1. M. Bellare, S. Micali: *How to Sign Given any Trapdoor Function*, Proceedings of Crypto '88, Springer Verlag LNCS series, pp. 200–215.
2. J. Benaloh, M. de Mare: *One-Way Accumulators: A Decentralized Alternative to Digital Signatures*, Proceedings of Eurocrypt '93, Springer Verlag LNCS series, pp. 274–285.
3. O. Goldreich: *Two Remarks Concerning the GMR Signature Scheme*, Proceedings of Crypto '86, Springer Verlag LNCS series, pp. 104–110.
4. J. Bos, D. Chaum: *Provably Unforgeable Signatures*, Proceedings of Crypto '92, Springer Verlag LNCS series, pp. 1–14.
5. D. Chaum, T. P. Pedersen: *Wallet Databases with Observers*, Proceedings of Crypto '92, Springer Verlag LNCS series, pp. 89–105.
6. R. Cramer, T. Pedersen: *Improved Privacy in Wallets with Observers*, Proceedings of Eurocrypt '93, Springer Verlag LNCS series, pp. 329–343.
7. R. Cramer, I. Damgård: *Secure Signature Schemes based on Interactive Protocols*, Proceedings of Crypto '95, Springer Verlag LNCS series, pp. 297–310.
8. R. Cramer, I. Damgård, T. Pedersen: *Efficient and Provable Security Amplifications*, Proceedings of 4th Cambridge Security Protocols Workshop, April 1996.
9. W. Diffie, M. Hellman: *New Directions in Cryptography*, IEEE Transactions on Information Theory IT-22 (6): 644–654, 1976.
10. C. Dwork, M. Naor: *An Efficient Existentially Unforgeable Signature Scheme and its Applications*, Proceedings of Crypto'94, Springer Verlag LNCS series, pp. 218–238.
11. T. ElGamal, *A Public-Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*, IEEE Transactions on Information Theory, IT-31 (4): 469–472, 1985.
12. A. Fiat, A. Shamir: *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Proceedings of Crypto '86, pp. 186–194
13. S. Goldwasser, S. Micali and R. Rivest: *A Digital Signature Scheme Secure Against Chosen Message Attacks*, SIAM Journal on Computing, 17(2): 281–308, 1988.
14. L. Guillou, J.J. Quisquater: *A Practical Zero-Knowledge Protocol fitted to Security Microprocessor Minimizing both Transmission and Memory*, Proceedings of Eurocrypt '88, Springer Verlag LNCS series, pp. 123–128.
15. G. H. Hardy, E. M. Wright: *An Introduction to the Theory of Numbers*, fifth edition, 1979, Oxford Science Publications.
16. *Information Technology - Security Techniques - Digital Signature Scheme Giving Message Recovery*, ISO/IEC Standard 9796, first edition, International Standards Organization, Geneva.
17. R. C. Merkle: *A Certified Digital Signature*, Proceedings of Crypto '89, Springer Verlag LNCS series, pp. 234–246.
18. M. Naor, M. Yung: *Universal One-Way Hash Functions and Their Cryptographic Applications*, Proceedings of 21st STOC, 1989, pp. 33–43.
19. National Institute of Technology and Standards: *Specifications for the Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication, US. Department of Commerce, 1993.

20. T. Okamoto: *Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes*, Proceedings of Crypto '92, Springer Verlag LNCS series, pp. 31–53.
21. B. Pfitzmann: *Fail-Stop Signatures Without Trees*, Hildesheimer Informatik-Berichte 16/94, Universität Hildesheim, Juni 1994.
22. R. Rivest, A. Shamir, L. Adleman: *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, Communications of ACM, 21 (1978), pp. 120 126.
23. J. Rompel: *One-Way Functions are Necessary and Sufficient for Secure Signatures*, Proceedings of 22nd STOC, 1990, pp. 387–394.
24. C. Schnorr: *Efficient Signature Generation by Smart Cards*, Journal of Cryptology, 4 (3): 161–174, 1991.