Cramer-Damgård Signatures Revisited: Efficient Flat-Tree Signatures Based on Factoring

Dario Catalano[1] and Rosario Gennaro[2]

[1] CNRS - École normale supérieure, Laboratoire d'informatique
45 rue d'Ulm, 75230 Paris Cedex 05, France.
`dario.catalano@ens.fr`

[2] I.B.M. T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598.
Email: `rosario@us.ibm.com`

Abstract. At Crypto 96 Cramer and Damgård proposed an efficient, tree-based, signature scheme that is provably secure against adaptive chosen message attacks under the assumption that inverting RSA is computationally infeasible.
In this paper we show how to modify their basic construction in order to achieve a scheme that is provably secure under the assumption that factoring large composites of a certain form is hard.
Interestingly our scheme is as efficient as the original Cramer Damgård solution while relying on a seemingly weaker intractability assumption.

1 Introduction

Digital Signatures are arguably the most important primitive of public-key cryptography [10]. Using digital signatures the receiver of a message can be assured that the message originated with a specific sender, and even more importantly, she will be able to prove such thing to a third party (non-repudiation). Because of the centrality of this concept it is very important to find signature schemes which are provably secure and efficient.

The concept of provable security for signature schemes (i.e. forgery should be equivalent to the solution of a well-defined conjectured hard problem) was formalized in the seminal paper by Goldwasser *et al.* [14] where an exact definition of what "forgery" means is given.

Starting from the scheme described in [14], several other provably secure signature schemes have been proposed in the literature that follows their paradigm. An important line of research has been to try to identify the minimal assumption needed to construct provably secure signature schemes. The assumption used in [14] was the existence of *trapdoor claw-free permutations*. Later, Bellare and Micali [2] showed that any trapdoor permutation would suffice. A breakthrough result came with Naor and Yung [18] who showed that it is possible to construct provably secure signatures out of *one-way permutations*, disposing of the trapdoor assumption which was considered essential. Finally Rompel [23] relaxed the assumption to the mere existence of one-way functions (which is easily seen to be the minimal assumption required).

However, the schemes mentioned above fall short in terms of their *efficiency* (which is measured as of computing time needed to produce and verify signatures and as of signature length). For example the original scheme in [14] builds a binary tree of height d, and the signature lenght and the computing time is $O(d)$. The parameter d is chosen so that 2^d is larger than the number of messages that the signer will ever sign.

It is thus important to research if using the properties of *specific* number-theoretic problems (like Factoring, RSA or Discrete Log) it is possible to devise provably secure yet efficient signature schemes.

For the case of the RSA function, Dwork and Naor [11] propose such a scheme, which was later improved by Cramer and Damgård [7]. The idea proposed in [11, 7] is to use specific properties of the RSA function to modify the original scheme in [14] to work with a "flat"tree, i.e. a tree with large branching factor $l > 2$. In the [11, 7] schemes, computation time and signature length are still $O(d)$ but now d is much smaller because all we need is that l^d is larger than the total number of signed messages.

At the same time Cramer [5] extended the basic GMR [14] technique to work with a flat-tree. The resulting scheme allows to obtain signatures that are somewhat shorter than GMR. This however comes at the cost of requiring much larger keys and public parameters: for example the signer is required to keep $l + 1$ different "trapdoors" and users of the scheme must agree on a common list of l random numbers (the latter is required also in [11]). In particular this means that the private storage for the signer is larger by a factor of l with respect to [11, 7]. The computational efficiency of Cramer's scheme [5] is comparable to [14], which is less efficient than [7].

Thus from a purely computational point of view (i.e. regardless of the assumption used), the method presented in [7] is more desirable since it uses less time and space. The open question, then, is to see if one can achieve the same efficiency as [7], only relying on a factoring assumption.

Our Contribution. In this paper we give a positive answer to this question, by showing how to construct an efficient flat-tree signature scheme whose security is based on the assumption that factoring large RSA moduli of a special form is hard. The restriction on the moduli N is that we require that the product of the smallest l primes divides $\phi(N)$. This restriction does not seem to affect the security of the factoring assumption, nor does it seem to make finding these moduli any harder.

Some components of our scheme (particularly the basic authentication step) are identical to the ones proposed by Cramer and Damgård in [7]. The security reduction to factoring is achieved by changing the key generation protocol and the choice of the public parameters. Because the basic authentication step remains the same, however, the efficiency of our scheme is pretty much equivalent to the efficiency of the scheme proposed in [7], while relying on a seemingly weaker assumption.

1.1 Other Related Work

Besides the works already mentioned in the Introduction, we point out that efficient provably secure signature schemes have been proposed using a variation on the RSA assumption. These works [13, 8] present efficient, state-free (all the above schemes, including ours, require the signer to keep some state) signatures based on a stronger assumption on the inversion of the RSA function. Although these schemes are more efficient than ours we stress that our goal was to prove the security of a reasonably efficient signature scheme based on the weaker assumption about factoring large integers, which subsumes both the regular and the strong RSA Assumptions.

A different approach followed in the literature is to try to prove "as much as possible" the security of efficient signature schemes like traditional RSA and schemes of the ElGamal family [12]. Starting from the work of Bellare and Rogaway [3] several papers proved that these schemes are secure (according to the [14] definition) in an idealized model of computation where a *random oracle* (an oracle that returns the result of a random function) is available to all parties. The random oracle is used to model "complicated hash functions" on which the security of the scheme relies. Although a proof in the random oracle model is better than no proof at all, it should not be automatically construed as a proof of security in the real model of computation. Indeed this is not the case, as proven in a result by Canetti *et al.* [4]. Since our scheme does not use a random oracle, we do not further discuss the random oracle model in this paper.

2 Definitions and Notations

We start with some definitions and notations. Given a probability space C we indicate with $x \leftarrow C$ the algorithm which assigns to x a random element according to C. In the case in which C is a finite set, $x \leftarrow C$ indicates the algorithm which assigns to x a random (uniformly chosen) element of C.

We say that a function $\epsilon(\cdot)$ is *negligible* if for every constant $c \geq 0$ there exists an integer k_c such that for all $k > k_c$ $\epsilon(k) < k^{-c}$

In the rest of the paper we assume that N is an n-bit composite modulus obtained as the product of two *Blum* primes p and q (i.e. p and q are such that $p \equiv q \equiv 3 \bmod 4$). We denote such moduli as *Blum modului*. We denote with $\lambda(N) = lcm(p-1, q-1)$. It is well known that for all $x \in Z_N^*$ we have that $x^{\lambda(N)} = 1 \bmod N$.

Consider now l (small) odd primes $\rho_1, \ldots, \rho_l$ and let σ be their product. We are going to consider Blum moduli N, such that $\forall\, i$ ρ_i is a divisor of $\lambda(N)$, but ρ_i^2 is not, and moreover $N >> \sigma^4$. Let's denote then with $BLUM(k, \rho_1, \rho_2, \ldots, \rho_l)$ the set of such Blum moduli with the property that $\lambda(N)/\sigma$ is of length k, i.e.

$$\begin{aligned} BLUM(k, \rho_1, \rho_2, \ldots, \rho_l) = \{ & N = pq \ : \ p, q \equiv 3 \bmod 4 \ , \\ & \rho_i | \lambda(N) \ , \ \rho_i^2 \not| \lambda(N) \ , \ |\lambda(N)/(\rho_1 \cdots \rho_l)| = k \} \end{aligned}$$

In the following we will assume that factoring such integers is hard even when given knowledge of the product σ of the small primes that divides $\lambda(N)$.

Assumption 1 (Factoring) *For every polynomial-time algorithm $\mathcal{A}$, and for every set of small primes $\rho_1, \ldots, \rho_l$, the following probability is negligible in k:*

$$Pr\left[\begin{array}{l} N \leftarrow BLUM(k, \rho_1, \rho_2, \ldots, \rho_l)\ , \\ \mathcal{A}(N, \rho_1, \ldots, \rho_l) = (p, q) \quad : \quad N = pq \end{array}\right]$$

Families of Hash Functions. We consider families of hash functions mapping strings of arbitrary length to strings of fixed length. Namely we consider a family $\mathcal{H} = \{\mathcal{H}_k\}_k$ where each $\mathcal{H}_k$ is a collection of functions of the form $H : \{0,1\}^* \rightarrow \{0,1\}^k$ for some integer k. $\mathcal{H}_k$ is polynomially samplable. We will be interested in hash functions that are *collision intractable*. A family $\mathcal{H}$ of hash functions is said to be collision intractable if it is infeasible to find two different inputs that map to the same output for a randomly chosen member of the family.

Definition 1 (Collision Intractability [9]). *We say that $\mathcal{H}$ is collision intractable if, for every probabilistic polynomial time algorithm $\mathcal{A}$ there exists a negligible function $\epsilon(\cdot)$ such that*

$$Pr[H \leftarrow \mathcal{H}_k;\ \mathcal{A}(H) = (x_1, x_2)\ s.t.\ x_1 \neq x_2\ and\ H(x_1) = H(x_2)] \leq \epsilon(k)$$

We now define digital signatures.

Definition 2 (Digital Signatures). *Let k be a security parameter, we define a digital signature as the triplet $(\mathcal{G}, \mathcal{SIG}, \mathcal{VER})$, where*

- *$\mathcal{G}$ is a polynomial time randomized algorithm that on input 1^k outputs a pair (PK, SK) of matching public and secret keys.*
- *$\mathcal{SIG}$ is the signing algorithm. It takes as input a message m, the keys PK, SK and possibly keeps some internal state. It produces as output a signature σ for m. This algorithm can be probabilistic.*
- *$\mathcal{VER}$ is the verification algorithm. It receives as input a message m, the public key PK and a signature σ, and checks if σ is valid according to m and PK. In other words $\mathcal{VER}(m, PK, \sigma) = 1$ if $\sigma = \mathcal{SIG}(m, PK, SK)$.*

The strongest notion of security for signature schemes was given by Goldwasser, Micali and Rivest [14]

Definition 3 (Secure signatures). *A signature scheme $(\mathcal{G}, \mathcal{SIG}, \mathcal{VER})$ is existentially unforgeable against an adaptive chosen message attack if it is computationally infeasible for a forger, who knows just the public key, to produce a valid signature σ on a message m even after having obtained polynomially many signatures on messages m_i of his choice from the signer.*

More formally, for every probabilistic polynomial time algorithm $\mathcal{F}$, there exists a negligible function $\epsilon(\cdot)$ such that

$$\Pr\left[\begin{array}{l} (PK, SK) \leftarrow \mathcal{G}(1^k) \\ for\ i = 1 \ldots n \\ \quad m_i \leftarrow \mathcal{F}(PK, m_1, \sigma_1, \ldots, m_{i-1}, \sigma_{i-1}) \\ \quad \sigma_i \leftarrow \mathcal{SIG}(m_i, PK, SK) \\ (m, \sigma) \leftarrow \mathcal{F}(PK, m_1, \sigma_1, \ldots m_n, \sigma_n); \\ m \neq m_i \ \ for\ \ i = 1 \ldots n, \ \ and\ \ \mathcal{VER}(m, PK, \sigma) = 1 \end{array}\right] \leq \epsilon(k)$$

3 The new scheme

Our scheme will make use of a l-ary tree (i.e. with branching degree l), which we call the *signature tree*. The root of the tree will be a random value S included in the public key of the signer. The tree has depth $d+1$ with a branching degree of l in the first d levels and a branching degree of 1 in the last level. By this setting we will allow the signer to sign up to l^d messages (we are going to assume l^d to be polynomial in k the security parameter). We now introduce some terminology: The first d levels of the tree are denoted as *expanding* levels, since every parent node S_j at level j ($j \in \{1, \ldots, d\}$) has l children (we have the root as $S_0 = S$). We call these nodes *expanding nodes*. The remaining level, level $d+1$, is called, the *terminal* level. Every parent node belonging to this levels has exactly one child. The parent nodes at the terminal level are denoted as *terminal nodes*. As usual, each terminal node's only child is called a leaf of the tree. We call an *item* a parent together with all his children and an *arc* a parent with one of his children. This means that every item has l arcs. A *path* from a node A to a node B is is the sequence of arcs that connects A with B.

Informally the signature algorithm will start "filling up" this tree. To sign the i^{th} message m_i, the signer will place m_i as the i^{th} leaf and will output an authentication chain that links m_i to the root of the tree (which is part of the public key). The verifier, will follow this authentication chain and if the end result matches the value in the public key, accepts the signature. Formal details follow.

3.1 Formal description on the scheme

We now give a formal detailed description of our scheme.

KEY GENERATION. The signer chooses l odd distinct primes[3] $\rho_i < 2^v$ (for some small enough parameter v) and sets $\hat{p} = \prod_{i=1}^{l/2} \rho_i$, $\hat{q} = \prod_{i=l/2+1}^{l} \rho_i$ and $\sigma = \hat{p}\hat{q}$. He then randomly picks two (distinct) large primes p' and q' of length $k/2$ such that $p = 2p'\hat{p}+1$ and $q = 2q'\hat{q}+1$ are two $(k+\omega)/2$-bit primes (for some parameter ω that depends on v and l). Then he sets $N = pq$ as the public modulus.

Note that by this position we have that N is a Blum integer such that ρ_i (but *not* ρ_i^2) divides $\lambda(N)$, and of the appropriate length. Notice also that 2 (but *not* $2^2 = 4$) divides $\lambda(N)$.

Denote with $E = 2\sigma = 2\rho_1 \cdots \rho_l$. The signer chooses uniformly and at random two E-th residues h, S in $\mathbb{Z}_N^*$ and a function H from a family of collision intractable hash functions. We will assume that H outputs a value in $\{0,1\}^\ell$, for some security parameter ℓ. For technical reasons, that will become apparent in the proof of security, the signer sets $e = 2^{\ell+1}$ and for each $i = 1, \ldots, l$ sets $e_i = \rho_i^{\ell_i}$, where ℓ_i is the minimum integer such that $e_i > 2^\ell$. The signer publishes $(N, h, S, e, e_1, \ldots, e_l, H, d)$, where d represents the depth of the tree, as his public

[3] The choice of these primes needs not satisfy any special requirement. For efficiency reason these primes could be chosen as the first l odd primes.

key and keeps private the factorization of the modulus. Note that this allows the signer to sign up to l^d messages.

Remark 1. The key generation algorithm is very similar to the one proposed by Naccache and Stern in [17]. They showed that the extra requirement on the choice of p, q in practice slows down the generation of N by around 9% with respect to the generation of a regular RSA modulus (see [17] for more details).

Signature Algorithm. The signer holds a tree of depth d with root S. All the nodes in the tree at the beginning are empty.

To sign the i^{th} message m_i the signer proceeds as follows:

1. He visits the path on the tree from the root to the i^{th} leaf, which is labeled with m_i. If a node j on this path has not been visited before, the signer labels it with a random E-residue S_j.
2. Let $(S, i_1, S_{i_1}, \ldots, i_d, S_{i_d})$ be the visited path (where each i_j is an index in $\{1, \ldots, l\}$). Then he solves the following equations
$$y_{i_1}^{e_{i_1}} = S \cdot h^{H(S_{i_1})} \bmod N$$
and for all $j = 2, \ldots, d$
$$y_{i_j}^{e_{i_j}} = S_{i_{j-1}} \cdot h^{H(S_{i_j})} \bmod N$$
To conclude the signature he computes a z_i such that
$$z_i^e = S_{i_d} \cdot h^{H(m_i)} \bmod N$$
3. The output signature on m_i is $\mathsf{sig}(m_i) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$.

Signature Verification. The receiver, given a message m, the public key $(N, h, S, e, e_1, \ldots, e_l, H, d)$ and a purported signature $\mathsf{sig}(m) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$, computes the following
$$S_{i_d} = z_i^e \cdot h^{-H(m_i)} \bmod N$$
followed by
$$S_{i_{j-1}} = y_{i_j}^{e_{i_j}} \cdot h^{-H(S_{i_j})} \bmod N$$
for all $j = d$ downto 1.

If the final value $S_0 \equiv S \bmod N$ the signature is accepted as valid.

Remark 2. Note that even though we perform iterated root extractions during the signing procedure we just need to assume that h and the S_j's above are E-th residues to make the above procedure work. Indeed we have that $gcd(e_i, \lambda(N)) = \rho_i$, so we can find α_i, β_i such that $\alpha_i e_i + \beta_i \lambda(N) = \rho_i$. This means that, in order to compute the $e_i = \rho_i^{\ell_i}$-th root of an E-residue x, the signer should first compute $\Delta = x^{\alpha_i}$ which by the above GCD computation is an e_i-root of x^{ρ_i} and then compute a ρ_i-root of Δ. A similar argument holds for e-roots.

Now let Δ be an E-residue and let δ_i one of its ρ_i-roots, i.e. $\delta_i^{\rho_i} = \Delta \bmod N$. In general the value δ_i can be computed in $O(\rho_i)$ time if we know the factorization of N (cf. [1]). Note that this is not a problem if one assumes that the primes ρ_i are all very small. However, if one wants to use slighly larger primes, the $O(\rho_i)$ solution may become too inefficient. In Appendix A we show a method to extract ρ_i-roots at the cost of a single modular exponentiation in Z_N^*.

The security of the scheme is stated in the following Theorem.

Theorem 1. *If Assumption 1 holds and H is a collision resistant hash function, then the digital signature scheme presented above is secure against an adaptive chosen message attack.*

The proof appears in the following Section 4. In the proof we use the following fact: Assume to have an algorithm $\mathcal{A}$ that on input N, e and an e-th residue y outputs an e^{th} root of y. In Appendix B we prove that it is then possible to construct a different algorithm $\mathcal{B}$, having black box access to $\mathcal{A}$, that factors the modulus with probability $1 - 1/e$.

Remark 3. Our presentation of the scheme, and consequently the theorem statement, assume the existence of collision-resistant hash functions. However it should be noted that factoring does imply the existence of collision-resistant hashing, thus we are not introducing any extra computational assumption. Moreover, using techniques similar to the ones presented in [7], one can completely dispense with the hash function H in our scheme. Either solution (implementing a factoring-based hashed function or changing the scheme so not to need one) would be however much more expensive than using, say, SHA-1. In order to keep the presentation simple, we decided to present the scheme this way, since we believe it is also conceptually clearer to "separate" the role of the hash function from the number-theoretic authentication step. In the final version of the paper we will show how to adapt the techniques in [7] to our scheme to avoid using H altogether.

4 Proof of Security

The proof goes by *reductio ad absurdum.* We assume that the proposed scheme is not secure, meaning that there exists an adversary $\mathcal{A}$ that can forge signatures with some non-negligible probability ϵ. Then we prove that if such an adversary exists, then it is possible to construct a probabilistic polynomial time algorithm $\mathcal{B}$ (a simulator) that, using $\mathcal{A}$ as an oracle, can factor with non negligible probability, thus contradicting the hypothesis of the theorem.

If we assume that such $\mathcal{A}$ exists, then his interaction with the signer would be as follows. First $\mathcal{A}$ gets the public key. Then for $i = 1, \ldots, t$ (where t is the maximum number of signatures the adversary is allowed to ask) he asks for the signature on a message m_i and receives back a valid signature $\mathsf{Sig}(m_i) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$. Then he will output $m \neq m_i$ and a valid signature $\mathsf{Sig}(m_j) = (z_j, y_{j_1}, j_1, \ldots, y_{j_d}, j_d)$ on it.

We argue that the public key and the verification tests on a valid signature imply that the forged signature must satisfy one of the following (mutually exclusive) conditions (where with S_{i_0} we denote S the root of the tree contained in the public key):

Type I For some $1 \leq i \leq t$, one has that $y_{j_k} = y_{i_k}$ for each $k = 1, \ldots, d$, $S_{i_d} = S_{j_d}$, but $z_j \neq z_i$.
Type II For some $1 \leq i \leq t$, there exist an index $1 \leq k' < d$ such that for all $k \leq k'$, $y_{j_k} = y_{i_k}$, $S_{i_{k'}} = S_{j_{k'}}$ but $y_{j_{k'+1}} \neq y_{i_{k'+1}}$.

If there is a forger that succeeds with non negligible probability, then there must be a forger that can successfully produce either a Type I forgery, or a Type II forgery with non negligible probability.

In the rest of the proof we will distinguish two cases, depending on the type of expected forgery. Since these two cases are exhaustive, one of them must happen with probability at least $\epsilon/2$.

Forgery of type I. The algorithm $\mathcal{B}$ (the simulator) is given as input a Blum modulus N of the appropriate form together with a set of l small primes $(\rho_1, \ldots \rho_l)$ such that for every ρ_i one has that $\rho_i | \lambda(N)$ but $\rho_i^2 \not| \lambda(N)$. We want to show how $\mathcal{B}$ can use the forgery received from $\mathcal{A}$ to factor N. Let t be the maximum number of sign-queries the adversary is allowed to ask (for simplicity we will assume that $\mathcal{A}$ will ask *exactly* t queries). The simulator generates his public key as follows. First it generates the public exponents $e, e_1, \ldots e_l$ as a real signer would do. Next, it sets $F = 2 \cdot e_1 \cdots e_l$, choses α, β uniformly and at random in $\mathbb{Z}_N^*$ and sets $h = \alpha^F \bmod N$ and $S = \beta^F \bmod N$. Notice that we can take e_i-roots of h, S (for any i) but not e-roots (since $e = 2^{\ell+1}$).

All the internal node, except those of depht d are computed in a similar way. The simulator sets $S_k = x_k^F \bmod N$ (where, once again, the x_k's are chosen randomly in $\mathbb{Z}_N^*$) and stores the x_k's for future usage. Observe that all the nodes generated this way – as well as S and h – are random E-residues in $\mathbb{Z}_N^*$, so they are distributed exactly as in the real signing process (more details below).

The simulator can generate valid signatures as follows. To sign the i-th message m_i, it chooses z_i at random in $\mathbb{Z}_N^*$ and sets

$$S_{i_d} = z_i^e h^{-H(m_i)} \bmod N$$

All the remaining relations can be easily computed as follows. For each index i_k in the path of the signature, the simulator sets

$$y_{i_k} = x_{i_{k-1}}^{F/e_{i_k}} (\alpha^{F/e_{i_k}})^{H(S_{i_k})} \bmod N$$

Finally it outputs the signature

$$\mathsf{Sig}(m_i) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$$

Observe that the signatures produced by the simulator are *perfectly* indistinguishable with respect to the signatures a real signer would generate. As a matter of fact the only difference between a real signature and a simulated one is the following. In the first case all the nodes of the tree – as well as the root S and the public value h – are E-residues (recall that $E = 2 \cdot \rho_1 \cdots \rho_l$), whereas in the simulation they are F-residues (with $F = 2 \cdot e_1 \cdots e_l$). However, since $e_i = \rho_i^{\ell_i}$ (for each index $i = 1, \ldots, l$) and N is a Blum modulus, every ρ_i-th residue is *also* an $\rho_i^{\ell_i}$ power. Consequently any E-residue is also an F-residue. Moreover, notice that, according to the simulation method described so far, the value α is never revealed to the adversary. In the terminal levels the (simulated) authentication procedure does not involve *any* e-root extraction. On the other hand, in the expanding levels, the authentication method requires the simulator to extract e_i-roots, but it is always the case that $e_i \neq e$. In other words the simulation is information-theoretically independent from α.

Now let $\mathsf{Sig}(m_j) = (z_j, y_{j_1}, j_1, \ldots, y_{j_d}, j_d)$ be the forgery produced by the adversary on a (up to now) unsigned message m_j. Since we are assuming the adversary creates a Type I forgery, for some previously produced signature $\mathsf{Sig}(m_i) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$ we have that $y_{j_k} = y_{i_k}$ for each $k = 1, \ldots, d$ but $z_j \neq z_i$.

This yields to the following system of equations:

$$(z_j)^e = S_{i_d} h^{H(m_j)} \bmod N$$

$$(z_i)^e = S_{i_d} h^{H(m_i)} \bmod N$$

Moreover, since H is collision resistant $m_i \neq m_j$ implies that $H(m_i) \neq H(m_j)$ and we can write $H(m_i) - H(m) = 2^\omega q$ for some $\omega \leq \ell$ and an odd $q \geq 1$.

From the two equations above we can compute

$$\left(\frac{z_j}{z_i}\right)^e = (h^q)^{2^\omega} \bmod N = (\alpha^{qF/2})^{2^{\omega+1}} \bmod N$$

Recall now that $e = 2^{\ell+1}$ so we get that

$$\left(\frac{z_j}{z_i}\right)^{2^{\ell+1-\omega}} = h^q \bmod N$$

Now, h^q has two square roots, of which we already know one: $\alpha^{qF/2}$. From the above equation we get that $(z_j z_i^{-1})^{2^{\ell-\omega}}$ is also a square root of h^q. Notice that $\ell - \omega \geq 0$ so we can easily compute the value without computing square-roots.

Observe that the adversary has no information at all regarding the original α chosen by the simulator (in an information theorethic sense). Consequently the value $(z_j z_i^{-1})^{2^{\ell-\omega}}$ is a square root of h^q that is different from $\alpha^{qF/2}$ with probability $1/2$. This immediately allows to factor the modulus.

Forgery of type II. The algorithm $\mathcal{B}$ is given as input a Blum modulus N of the appropriate form together with a set of l small primes $(\rho_1, \ldots \rho_l)$ such that for every ρ_i one has that $\rho_i | \lambda(N)$ but $\rho_i^2 \not| \lambda(N)$.

The simulator starts generating the signing public key by choosing a random index $1 \leq \delta \leq l$. This random choice can be interpreted as the simulator "guessing" the value of $j_{k'+1}$, the index of the first child where the forgery and the regular signature path of the tree will differ.

Next it creates the public exponents $e, e_1, \ldots e_l$ as prescribed by the key generation algorithm. Then it chooses a random element $\alpha \in \mathbb{Z}_N^*$, computes $G = e \cdot e_1 \cdots e_{\delta-1} \cdot \rho_\delta \cdot e_{\delta+1} \cdots e_\tau$ and sets $h = \alpha^G \bmod N$.

The simulation proceeds by letting $\mathcal{B}$ precompute the authentication tree in order to be able to produce t valid signatures. This precomputation phase goes very similarly to the one described before. The main difference here is that the root and the internal nodes of the tree are computed in a bottom-up fashion (rather than top-down, as for the forgeries of type one).

For each node S_{i_d} (nodes of depth d) the simulator chooses a random element x_{i_d} and sets $S_{i_d} = x_{i_d}^G \bmod N$. Once the nodes of level d are prepared, one can construct the expanding nodes, item by item.

Here, for simplicity, we show the method for a generic item I. The basic idea is to construct the parent node S_{I_0} in terms of its δ-th child S_{I_δ}. In particular the simulator chooses a random value $x_I \in \mathbb{Z}_N^*$, sets

$$S_{I_0} = x_I^{G \cdot \rho_\delta^{\ell_\delta - 1}} h^{-H(S_{I_\delta})} \bmod N$$

and stores the values S_{I_0} and x_I (in the following, for each item I, we will refer to x_I as to the *basis* of S_{I_0}).

Using this methodology the simulator can (inductively) generate the entire tree. Each new level is obtained by combining the items of the previous level in a tree structure (the roots of the items of level k play the role of the leaves to construct the items of level $k-1$). At the end of this phase the simulator comes up with a global root S, which is included as part of the public key.

On top of this construction to sign the message m_i, the simulator does as follows. First he computes the path $(i_1, \ldots, i_d)$ from the root to the i^{th} leaf of the tree. Then he proceeds according to the following procedure:

for $k = 1$ **to** d
  Assume S_{i_k} is the b-th child of $S_{i_{k-1}}$
  Let x_{i_k} be the basis of $S_{i_{k-1}}$
  if $b == \delta$
    Set $y_{i_k} = x_{i_k}^{G/\rho_\delta}$
  if $b \neq \delta$
    Set $y_{i_k} = x_{i_k}^{G/e_b \cdot \rho_\delta^{\ell_\delta - 1}} \cdot (\alpha^{G/e_b})^{H(S_{i_k})}$
Set $z_i = x_{i_d}^{G/e} (\alpha^{G/e})^{H(m_i)}$
Output the signature $\mathsf{Sig}(m_i) = (z_i, y_{i_1}, i_1, \ldots, y_{i_d}, i_d)$

In other words, the adversary easily computes e-roots and e_i-roots (for $i \neq \delta$) because all the values are G-residues and he knows G-roots of them. For the case $i = \delta$ it is not necessary to compute e_δ-roots thanks to the way in which the internal nodes have been prepared.

If the adversary produces a valid forgery $\mathsf{Sig}(m_j) = (z_j, y_{j_1}, j_1, \ldots, y_{j_d}, j_d)$, one can "use" it to break Assumption 1 as follows. Since we are dealing with a forgery of the second type, there exists and index k (such that $1 \leq k \leq d$) for which one has that $S_{i_{k-1}} = S_{j_{k-1}}$ but $y_{i_k} \neq y_{j_k}$. Moreover, since $\mathcal{B}$ simulates a real signer perfectly, with probability $1/l$ one has that S_{j_k} is the δ-th child of $S_{j_{k-1}}$. If this is the case we can then consider the following equations:

$$y_{j_k}^{e_\delta} = S_{i_{k-1}} h^{H(S_{j_k})} \bmod N$$

$$y_{i_k}^{e_\delta} = S_{i_{k-1}} h^{H(S_{i_k})} \bmod N$$

which dividing term by term become

$$Y^{e_\delta} = h^{\Delta H} \bmod N$$

where we set $Y = (y_{j_k}/y_{i_k})$ and $\Delta H = H(S_{j_k}) - H(S_{i_k})$.

Once again since H is collision resistant, from the fact that $S_{j_k} \neq S_{i_k}$ we can assume that $\Delta H \neq 0$. Therefor we can write $\Delta H = \rho_\delta^\omega q$ with $q \geq 1$, such that $\gcd(q, \rho_\delta) = 1$. Moreover $\omega < \ell_\delta$, because of the way we chose ℓ_δ.

The above equation can then be rewritten as

$$Y^{\rho_\delta^{\ell_\delta}} = \left(\alpha^{\frac{qG}{\rho_\delta}}\right)^{\rho_\delta^{\omega+1}} \bmod N$$

which implies that the value $Z = Y^{\rho_\delta^{\ell_\delta - \omega - 1}}$ is an ρ_δ root of h^q, that is different with respect to $\alpha^{\frac{qG}{\rho_\delta}}$ with probability $1 - 1/\rho_\delta$. Again notice that $\ell_\delta - \omega - 1 \geq 0$ so the value Z can be easily computed without computing ρ_δ-roots.

5 Security Analysis

For lack of space we cannot discuss in more details our intractability assumption. In the full version of this paper we give some evidence why assuming $N >> \sigma^4$ seems to be safe. We point out here, however, that the same analysis was already presented in [17]. The interested reader is referred to [17] for details.

5.1 Comparison with GMR

In [14] Goldwasser, Micali and Rivest proposed the first example of digital signature scheme secure against adaptive chosen message attack. The scheme relies on the existence of *claw free* permutations, but the authors propose a concrete implementation based on the hardness of factoring. The reader is referred to [14] for the technical details; here we compare the practical performance of our scheme with respect to the one presented in [14].

Their scheme is based on a binary tree. As we mentioned before the depth of the tree is $\hat{d} = \log K$. Let us denote with $\delta > 1$ the ratio $\hat{d}/d$.

The length of the signature is about $2\hat{d}n$ bits, i.e. $2n$ bits per level of the tree. Notice that this is a factor of 2δ longer than our signatures.

The basic authentication step, performed at each level of the tree, consists of taking repeated square roots. In the original scheme in [14] the number of square roots taken at each level is about $2n$, where n is the length of the modulus. This happens because the number of square roots taken is proportional to the length of the information being authenticated. However to obtain a fair comparison with our scheme, we should improve the scheme in [14] by introducing a separate collision-resistant hash function H, like we did in our scheme. If one hashes the information at each step, before applying the authentication step, we reduce the work to 2ℓ square-root computations per level of the tree. By using the speed-up trick suggested by Goldreich (cf. Section 10.2 of [14]) this is equivalent to one exponentiation with an ℓ bit exponent, and one full exponentiation $\bmod N$, per level of the tree, i.e. roughly $1.5(\ell+n)$ multiplications. Thus the worst-case cost of computing a signature is $1.5\hat{d}(\ell+n)$ multiplications, which is a factor δ slower than ours.

To compute the amortized complexity of signatures in [14] we need to multiply the cost of the basic authentication step, by $2^{\hat{d}}$ (the number of nodes divided by two) [4] and then divide by $2^{\hat{d}}$ (the number of signatures). The net result is that the amortized cost is $1.5(\ell + n)$ multiplications per signature, the same as ours.

Similarly the verification of a signature requires the computation of about 2ℓ squarings at each level of the tree, for a total of $2\ell\hat{d}$ multiplications. Verification in [14] is thus a factor of $2\delta/3$ slower than in ours.

Let us consider a specific example in which $n = 1024$, $d = 80$, $l = 32$ (i.e. $\hat{d} = 16$) and $\ell = 160$. In this case $\delta = 5$ and we immediately obtain that our signatures are a factor of 10 shorter than the ones in [14]. The worst case complexity of computing a signature is also 5 times smaller in our scheme, while the amortized complexity is the same. Finally verification time is about three times as fast in our scheme.

5.2 Comparison with Cramer-Damgård

It is not hard to see that our scheme is very similar to the scheme proposed by Cramer and Damgård in [7]. Thus the efficiency of our scheme is identical to the one of the scheme proposed there, while relying on a weaker assumption.

6 Conclusions

We presented a new and efficient signature scheme, which is provably secure against adaptive chosen message attack under the assumption that factoring large composites of a certain form is infeasible.

[4] This is because a basic authentication step in [14] requires to authenticate an entire (binary) item.

Our scheme shows that the "flat-tree" approach can lead also to efficient signatures under a factoring assumption, while previous proposals relied either on the seemingly stronger RSA Assumption or were less efficient.

In terms of efficiency our scheme is equivalent to the RSA-based scheme presented in [7], and much better than the factoring-based ones in [14] and in [5].

Acknowledgements. We thank Pascal Paillier for helpful discussions.

References

1. E. Bach and J. Shallit. Algorithmic Number Theory. Vol.1 Efficient Algorithms. *MIT Press.* 1996.
2. M. Bellare and S. Micali How to sign given any trapdoor permutation *Journal of the ACM* no. 39(1), pages 214-233, 1992
3. M. Bellare and P. Rogaway Random Oracles are Practical: A paradigm for designing efficient protocols. *Proc. of First ACM Conference on Computer and Communications Security*, pages 62-73, 1993
4. R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. *Proc. 30th ACM Symposium on Theory of Computing*, 1998
5. R. Cramer Modular design of secure yet practical cryptographic protocols. Ph.D. Thesis, University of Amsterdam, 1996.
6. R. Cramer and I. Damgård. Secure signature schemes based on interactive protocols. *Proc. of Crypto '95* LNCS no. 963, pp.297-310.
7. R. Cramer and I. Damgård. New Generation of Secure and Practical RSA-based signatures. *Proc. of Crypto '96* LNCS no. 1109, pages 173-185.
8. R. Cramer and V. Shoup. Signature schemes based on the Strong RSA assumption. *Proc. of 6th ACM Conference on Computer and Communication Security 1999.*
9. I. Damgård. Collision free hash functions and public key signature schemes. *Proc. of Eurocrypt '87* LNCS no. 304, pages 203-216.
10. W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644-654, November 1976.
11. C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *J. of Cryptology* 11(3) 1998, pages 187-208.
12. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *Proc. of Cryypto '84* LNCS no. 196, pages 10-18.
13. R. Gennaro, S. Halevi and T. Rabin. Secure Hash-and-Sign Signatures Without the Random Oracle. *Proc. of Eurocrypt '99* LNCS no. 1592, pages 123-139.
14. S. Goldwasser, S. Micali and R. Rivest. A digital signature scheme secure against adaptive chosen message attacks. *SIAM J. on Computing* 17(2):281-308 1988.
15. N. Koblitz *A course in number theory and cryptography*, 2nd ed., Springer Verlag
16. R. Merkle. A Digital Signature based on a Conventional Encryption Function. *Advances in Cryptology–Crypto'87.* LNCS, vol.293, pp. 369–378, Springer–Verlag, 1988.
17. D. Naccache and J. Stern. A new cryptosystem based on higher residues. *Proc. of the 5th ACM conference on on computer and communication security, ACM press (1998), pp.59-66.*
18. M. Naor, M. Yung. Universal one-way hash functions and their cryptographic applications *Proc. of* 21st *ACM STOC* pages 33-43, 1989.

19. B. Pfitzmann. Digital Signatures Schemes - General Framework and Fail-Stop Signatures. Lecture Notes in Computer Science no. 1100 Springer.
20. D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *J. of Cryptology.* 13(3):361–396. Springer. Summer 2000.
21. M. Rabin. Digital Signatures and Public Key Encryptions as Intractable as Factorization. MIT Technical Report no. 212, 1979
22. R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126
23. J. Rompel. One-way functions are necessary and sufficient for secure signatures. *Proc. of 22nd STOC* 1990, pages 387-394.

A Efficient Root Extractions

With the following lemma we show a simple method (taking advantage of the fact that the ρ_i's are all odd primes) to extract ρ_i-roots in a (asyntotically) more efficient way.

Lemma 1. *Let p be a Blum prime of size k. Let e be a prime such that $e|p-1$ but $e^2 \not| p-1$. Then there exists an efficient algorithm, taking as input an e-residue a, that returns as output an e-root of a in time $O(k^3)$.*

Proof. First note that the prime p can be written as $p = 2em + 1$ where m is an odd integer such that $\gcd(e, m) = 1$. Since a is an e-residue in $\mathbb{Z}_p^*$ it must be true that

$$a^{\frac{p-1}{e}} \equiv 1 \bmod p$$

Now let B such that $\frac{p-1}{e} + B = Ae$ for some A over the integers. The equation above can the be rewritten as

$$a^{\frac{p-1}{e}} \cdot a^B \equiv a^{Ae} \bmod p$$

or better

$$a^{Ae} \equiv a^B \bmod p$$

Furthemore observe that since $\gcd(2m, e) = 1$ it has to be the case that $\gcd(B, e) = 1$. This means that, using the extended Euclidean algorithm, it is possible to compute two values λ and μ such that $\lambda B + \mu e = 1$ over the integers. Thus the equation above becomes

$$a^{\lambda B + \mu e} \equiv (a^A)^{e\lambda} \cdot a^{\mu e} \bmod p$$

and then

$$a \equiv \left(a^{A\lambda+\mu}\right)^e \bmod p$$

Thus $a^{A\lambda+\mu}$ is an e-root of a.

The cost of the described method is dominated by the cost of the Extended Euclidean Algorithm which requires $O(k^3)$ bit operations.

B Two Simple Lemmas

The following two lemmas are invoked during the proof of security of the signature scheme.

Lemma 2. *Let $N = pq$ be the product of two primes. Let e be a divisor of $\lambda(N)$ with multiplicity one (i.e. e^2 does not divide $\lambda(N)$) such that e divides either $p-1$ or $q-1$ but not both of them. Then every e-th residue has exactly e different e-th roots.*

Proof. It is a well known fact from number theory [15] that in every finite cyclic group G, the equation $x^d = a$ has $gcd(d, ord(G))$ different solutions. This fact, however, cannot be immediately applied to Z_N^* because it is not a cyclic group, but can be applied to the cyclic groups Z_q^* and Z_p^* having order, respectively, $\phi(q) = (q-1)$ and $\phi(p) = (p-1)$ (see [15] for details).
Without loss of generality assume that e divides $p-1$ but does not divide $q-1$. Now from the equation $y = x^e \bmod N$, we derive the equations

$$y = x^e \bmod p \tag{1}$$

and

$$y = x^e \bmod q \tag{2}$$

Equation 1 has then $gcd(e, (p-1)) = e$ different solutions and equation 2 has $gcd(e, (q-1)) = 1$ different solutions. Using the Chinese Remainder Theorem [15], these can be combined to yield e different solutions modulo N.

Lemma 3. *Let $N = pq$ be the product of two primes. Let e be a divisor of $p-1$ (resp. $q-1$) but not a divisor of $q-1$ (resp. $p-1$) with multiplicity one. Let a be an e-residue in Z_N^* and y_1, y_2 two distinct solutions of the equation $x^e = a \bmod N$. Then there is an efficient algorithm that on input y_1 and y_2 returns the factorization of N.*

Proof. Without loss of generality assume that e divides $p-1$. Since the equation $x^e = a \bmod q$ has only one solution, it must be the case that

$$y_1 \equiv y_2 \bmod q \tag{3}$$

On the other hand since $y_1 \neq y_2 \bmod N$ it has to be the case that

$$y_1 \not\equiv y_2 \bmod p \tag{4}$$

Equation 3 tells us that $y_1 - y_2 \equiv 0 \bmod q$ and thus, since $y_1, y_2 < N$, $gcd(y_1 - y_2, N)$ is a non trivial factor of N.

The two lemmas above have the following consequence. Assume to have an algorithm $\mathcal{A}$ that on input N, e and an e-th residue y outputs an eth root of y. From the lemmas above it is immediate to see that is then possible to construct a different algorithm $\mathcal{B}$, having black box access to $\mathcal{A}$, that factors the modulus with probability $1 - 1/e$ (just feed $\mathcal{A}$ with $y = x^e \bmod N$, where x is chosen randomly, and with probability $1 - 1/e$ $\mathcal{A}$ will return a root different than x).