An Efficient Existentially Unforgeable Signature Scheme and its Applications*

Cynthia Dwork † Moni Naor ‡

Abstract

A signature scheme is *existentially unforgeable* if, given any polynomial (in the security parameter) number of pairs

$$(m_1, S(m_1)), (m_2, S(m_2)), \ldots (m_k, S(m_k))$$

where $S(m)$ denotes the signature on the message m, it is computationally infeasible to generate a pair $(m_{k+1}, S(m_{k+1}))$ for any message $m_{k+1} \notin \{m_1, \ldots m_k\}$. We present an existentially unforgeable signature scheme that for a reasonable setting of parameters requires at most 6 times the amount of time needed to generate a signature using "plain" RSA (which is *not* existentially unforgeable). We point out applications where our scheme is desirable.

*Preliminary version appeared in Crypto'94

†IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120. Research supported by a BSF Grant 32-00032-1. E-mail: dwork@almaden.ibm.com.

‡Incumbent of the Morris and Rose Goldman Career Development Chair, Dept. of Applied Mathematics and Computer Science, Weizmann Institute of Science, Rehovot 76100, Israel. Some of this work performed while at the IBM Almaden Research Center. Research supported by a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences and a BSF Grant 32-00032-1. E-mail: naor@wisdom.weizmann.ac.il.

1 Introduction

A digital signature, just like a handwritten one, is a method that allows one party to "convince" another party that a third party indeed "approved" a given message. While this intuition is appealing, in order for it to make sense we must specify what "convince" and "approved" mean. The notion we concentrate on in this paper is that of existential unforgeability. A signature scheme is *existentially unforgeable* if, given a sequence of pairs

$$(m_1, S(m_1)), (m_2, S(m_2)), \ldots (m_k, S(m_k))$$

where $S(m)$ denotes the signature on the message m, it is computationally infeasible to generate a pair $(m_{k+1}, S(m_{k+1}))$ for any message $m_{k+1} \notin \{m_1, \ldots m_k\}$. While this definition may seem excessively demanding of the signature scheme, since it does not permit forgeries even on "nonsensical" messages, we are not aware of any other way to obtain a robust notion of of security. Furthermore, as the discussion below shows, there are "real life" situations where this is precisely the security needed.

Consider the problem of providing a "receipt" for data stored in a *document repository*, where the data can be of arbitrary form, much as one is provided with a claim check at a left luggage counter. In the most simple implementation, the receipt would just be a pair, consisting of an identifier and a signature on this identifier. If the signature scheme is existentially *forgeable*, then any party can produce such a pair, without knowing the secret signing key. An *existentially unforgeable* signature scheme prevents this. In such a scheme any signed document, nonsensical or otherwise, has necessarily been signed by the claimed signer. Assume that both the document owner and the repository employ existentially unforgeable digital signature schemes. The document owner signs the document to be stored; the document repository issues a signed receipt for the signed document. Existential unforgeability of the repository's signature ensures that any claimed receipt is indeed a valid receipt. Existential unforgeability of the document owner's signature ensures that the document repository cannot return a different document than the one that was stored, not even a nonsensical, but different, document.

Our own interest in finding efficient existentially unforgeable signature schemes comes from the problem of signing faxed documents. Since faxed documents have received legal standing in court, it is essential to use a signature scheme appropriate to this environment. Addressed in detail in Section 5, we briefly describe this environment and explain why it requires existentially unforgeable signatures. We consider a scenario in which a FAX is being sent by a machine equipped with a storage device, but is being received by a more ordinary FAX machine. Both can perform on-line calculations, but only the sender can store large quantities of data. Let D be a string representing a document that is to be signed; for example, D may be the result of scanning a paper document, or it could be a PostScript file. Let h be a collision-intractable hash function[1], and let S be a signing function. Let us assume the sender sends to the receiver the triple $(D, h(D), S(h(D)))$. Even if the receiver can check that the document has been correctly hashed and that the signature on the hash is valid, once the document represented by D is printed out on the receiver's FAX machine, there is no way to re-capture D from the printed image. For example, scanning the printed

[1] A hash function is collision intractable if it is computationally infeasible to find $x \neq y$ such that $h(x) = h(y)$; collision-intractable hash functions are discussed further in Section 5.

image optically will almost surely produce some string $D' \neq D$, since the scanned image may be slightly tilted, or dirty, *etc.* Since by assumption the receiver cannot store the string D, without some additional machinery the signature cannot provide irrefutability: the receiver cannot prove to a third party that D was received from the sender because, in order to do that, D must be exhibited at the time of proof. Since the receiver cannot store D, the receiver cannot exhibit D at a later time. Thus, if the receiver has no means of storing the string D itself (not just printing the document), such as a disk or tape drive, then some other party must store the data and issue a receipt to the receiver. The natural, although somewhat improbable, party to do this is the not necessarily trustworthy sender. Ideally, the receipt would be the pair $(h(D), S(h(D)))$, the explicit understanding being that if the receiver produces such a receipt, then the sender is obliged to produce a document hashing to this value. The problem with using an existentially *forgeable* signature scheme here is that, since $h(D)$ "looks random," anyone could generate what appears to be a valid receipt, just as in the claim check example. Thus, it is essential that the signature scheme be existentially unforgeable. With such a scheme, if the receiver produces $(h(D), S(h(D)))$, then necessarily a document hashing to $h(D)$ was signed by the sender.

In this paper we present an efficient existentially unforgeable signature scheme which we believe is the first practical one. For a reasonable choice of parameters it is about 30 times more efficient than the best previous proposals [14, 20, 18]. The security of the scheme relies only on the following *RSA assumption:*

Assumption 1 *Let N be the product of two large primes. Then, without knowing the factorization of N, it is computationally infeasible to extract pth roots mod N, where p is a random prime.*

A slightly stronger (and quite common) version of this assumption states that it is hard to extract pth roots mod$N$ *for small* primes p. Such an assumption makes the verification of signatures more efficient. A precise formulation of the assumptions is provided in Section 4.

We compare the cost of our scheme to that of "plain" RSA, *i.e.*, where the signature on a message m is $m^{1/p} \bmod N$. For a reasonable choice of parameters, the cost of signing and verifying in our scheme is at most six times that of RSA. In fact, the *amortized* cost of signing is only twice that of plain RSA; that is, over the lifetime of the system the amount of work spent on signing is at most twice that which would have been spent using plain RSA. Thus, in almost every scenario in which it is feasible to apply RSA it should be feasible to use our scheme (particularly if the signature generation is not the computational bottleneck).

The remainder of this paper is organized as follows. In in the next section we summarize the history of digital signatures, emphasizing work relevant to our scheme. In Section 3 we describe our scheme. The proof of security of our scheme appears in Section 4. In Section 5 we describe how to use the proposed scheme (or any other existentially unforgeable scheme) in the context of signing faxes.

2 Related Work and Its Influence

Since the introduction of the concept of digital signatures by Diffie and Hellman [11] and the first proposals of candidates for implementation [23, 30], the subject has been widely studied. In this section we briefly summarize the major developments (not necessarily in chronological order), especially those pertaining to the scheme proposed in this paper.

Goldwasser, Micali, and Rivest [20] formalized the notion of security of a signature scheme. They separated the description of the types of attack that a system might suffer from the definition of what it means to break the system. The strongest form of security they defined is *existential unforgeability under an adaptive chosen plaintext attack*. In an adaptive chosen plaintext attack, the adversary (or would-be forger), with the cooperation of the signature algorithm, obtains signatures on messages of the attacker's choice, chosen adaptively. The scheme is *existentially forgeable* if, after the attack, the forger is able to produce a valid signature on even one previously unsigned message, *without* the cooperation of the signature algorithm. (See exact definition in [20]; the definitions used here are the same.) It is *existentially unforgeable* if the adversary cannot sign even one, possibly nonsensical, new message.

Existential unforgeability under adaptive chosen message attack was considered too stringent a requirement for practical purposes. Firstly, the chosen plaintext attack is very generous to the forger. However, this attack captures the fact that there is no practical way to restrict the types of messages a user might ever wish to sign. Moreover, many security systems use the ability to sign random challenge messages as a proof of identity (see, *e.g.*, [26]), creating a perfect environment for a chosen plaintext attack. Secondly, existential unforgeability seemed an excessive requirement, with the following intuition. If, in an application, the signature function is to be applied to whole (rather than hashed) messages, then, it was argued, it should be enough to preclude the adversary from signing a message of her choice; it should not matter if the adversary can produce signatures on nonsense messages. If, instead, the signature function is to be applied to a collision-intractable hash of the message, then the adversary that could produce a signature on some arbitrary string s would not be able to find a message that hashes to s, and therefore would not be able to commit any meaningful forgery. However, as seen from the claim check and FAX examples outlined in the Introduction, there are indeed real-world applications for existential unforgeability. The RSA [30] and Rabin [28] schemes are known *not* to have this desirable property. Note that common ways of applying these schemes, involving signing $h(m)$ where h is some hash function with mysterious powers, are not known to be existentially forgeable; however, no proof of existential unforgeability for such a scheme is known (unless one resorts to such assumptions as the existence of a publicly accessible (publicly computable) truly random function (see, *e.g.*, [4])).

The implementation of an existentially unforgeable signature scheme suggested in [20] was based on the hardness of factoring and various improvements were suggested in [18]. Constructions based on more general assumptions (trapdoor permutations, one-to-one one-way functions and one-way functions) were given in [3, 25, 31]. These schemes are all rather inefficient in that they employ a tree whose height is proportional to the logarithm (to a *small* base) of the total number of messages signed by the system. Signing and verifying both involve tracing a path from the root to a leaf, where moving from node to node is quite expensive. For instance, it is comparable to two RSA computations in [20], as optimized in [18].

Our scheme employs a tree as well. However it is fat and, consequently, very shallow – a logarithm to a *large* base of the total number of messages signed by the system. This means that in a lifetime of use the tree is very unlikely to need more than three levels.

There are several constructions of one-time or fixed-time signature schemes that are existentially unforgeable. (One-time means that the public key is good for one signature only; fixed-time means that there is an *a priori* upper bound on the number of messages the scheme can sign. The size of the public key is usually related to this number.) The first such scheme is due to Lamport (described in [11] and used in [14, 22, 25, 31]). This scheme requires as many invocations of a one-way function as there are bits to be signed (some improvements are known). The scheme of Bos and Chaum [7] can be viewed as a fixed-time signature scheme. In this scheme the size of the public information needed grows at least as fast as the square root of the number of messages the scheme should be able to sign.

Even, Goldreich, and Micali [14] tried to combat the computational cost of signature schemes by distinguishing between *on-line* and *off-line* computation. Their scheme requires extensive pre-computation, "between" signing of different documents, but the on-line computation required for signing is very efficient. The size of a signature is rather large.

The El-Gamal scheme [13] signature scheme relies on no cleanly specified function; moreover, given a legitimately signed document in that scheme, it is possible to generate other legitimate signatures and messages; that is, the scheme is not existentially unforgeable. (For recent results concerning the security of the El-Gamal scheme see [6, 27].) The Fiat-Shamir scheme [16] and its descendants [17, 24, 32] are very efficient, since, unlike RSA and related schemes, they do not require modular exponentiation. However, they do require that the "one-way hash" function actually be something stronger, more like a black-box random function (no precise definition of the assumptions needed is given). None of these schemes is known to be existentially unforgeable.

Most of the ingredients of our scheme have appeared before; the contribution of this work is merely in finding the right mixture that makes the full scheme efficient. The idea of using exponentiation to hide information appears in the original RSA signature scheme [30]. Fiat and Shamir employ the subset product technique for signing [16]. Merkle [22] suggested the tree authentication scheme (also used in [25]), but in his scheme the tree cannot be shallow. The scheme in [7] is similar in spirit to the one-time version of our scheme used in every node. Bellare and Micali [3] suggested a tree-based scheme where nodes are "revived" by choosing a new trapdoor permutation which, in turn, is authenticated by the parent of the current node. Our scheme can be seen as an efficient way of performing this, by replacing the trapdoors of [3] with "masks" from the Fiat-Shamir scheme [16].

3 The Scheme

3.1 Outline

In rough outline, the scheme works as follows. Every signer s has, as in any signature scheme, a pair of keys. The *public* key is used to verify the signature on messages reputedly signed by s. The *private* key contains information known only to s, and is used by s to compute signatures on messages of the signer's choice.

The signer maintains a short, very bushy tree. Each message (or message digest) signed will be stored at its own leaf of the tree. There are two parameters, ℓ and k. The outdegree of the tree is k. For concreteness, in this informal description, we take $k = \ell = 1000$. In this case, if the tree is of height 3 then it has a billion leaves and can sign up to billion messages (it is possible to extend the scheme on the fly to sign any number of messages). Assume also, for this discussion, that all messages have length at most one thousand bits (larger messages may be hashed down to this length or broken into pieces, with each piece signed individually; we defer this discussion to Section 5).

Although the tree will be large, the scheme ensures that at any time the signer need only maintain information stored in a single root-leaf path. The nodes of the tree are accessed in a depth-first left-right manner. Thus, the leaves are accessed from left to right: the first message to be signed is stored at the leftmost leaf, the second at the next-to-leftmost leaf, and so on.

A signer's public key is an integer N, which is the product of two large primes, and a random 1000-bit string y_{root}. We remark that y_{root} can be the same for all signers, as long as it is initially chosen as a random ℓ-bit number. y_{root} is implicitly stored at the root of the signer's tree. Let us assume that $J \geq 0$ messages have been signed and that the signer wishes to sign the $(J+1)$st message, say, m. Let w be the $(J+1)$st leftmost leaf and let $\pi = (v_0 = \mathit{root}, v_1, \ldots, v_d)$ be the path from the root to the leaf $v_d = w$ (d is the depth of w; we have taken $d = 3$ for this discussion). For each step $i = 1, \ldots, d$ in the path π, if v_i has not previously been accessed (that is, v_i is not on the path from the root to any of the leftmost J leaves), then

1. if $i < d$, then the signer chooses a random 1000-bit label ξ (this is the information "stored" at v_i) and *authenticates* this string using the label of the parent of v_i; the signer stores ξ and its *authenticator* (described below) at v_i;

2. if $i = d$ (that is, if $v_i = v_d = w$), then the signer authenticates m using the label of the parent of w and stores its authenticator at w.

The signature on m is the path π together with the information stored at the nodes of π.

Intuitively, the scheme is practical because we "re-use" an internal node many times – once for each leaf in the subtree rooted at this node.

The basic authentication step requires two lists, which are common to all signers (and known to all verifiers): P, containing one thousand primes, and X, containing one thousand random 1000-bit strings. Let u be an internal node in the tree, let v be the jth child of u and let y_u and y_v be their respective labels. The goal of a *basic authentication step* is to authenticate y_v using y_u (which is assumed to have been authenticated already) and the jth prime in the list P. This is done as follows: the bits of y_v are used to select a subset of the elements of X; then the product of y_u and the selected elements is computed. Finally, the authenticator is the p_jth root (modulo N) of this product. Note that since the labels of the internal nodes are all random, there is virtually no chance that any two such labels are identical. Thus, we never use the same (y_u, p_j) pair twice in a basic authentication step. This is critical in obtaining existential unforgeability.

3.2 Detailed Description

The scheme is parameterized by ℓ and k and we consider ℓ to be the security parameter of the scheme. The scheme works with numbers (labels) of length ℓ; thus ℓ should be chosen

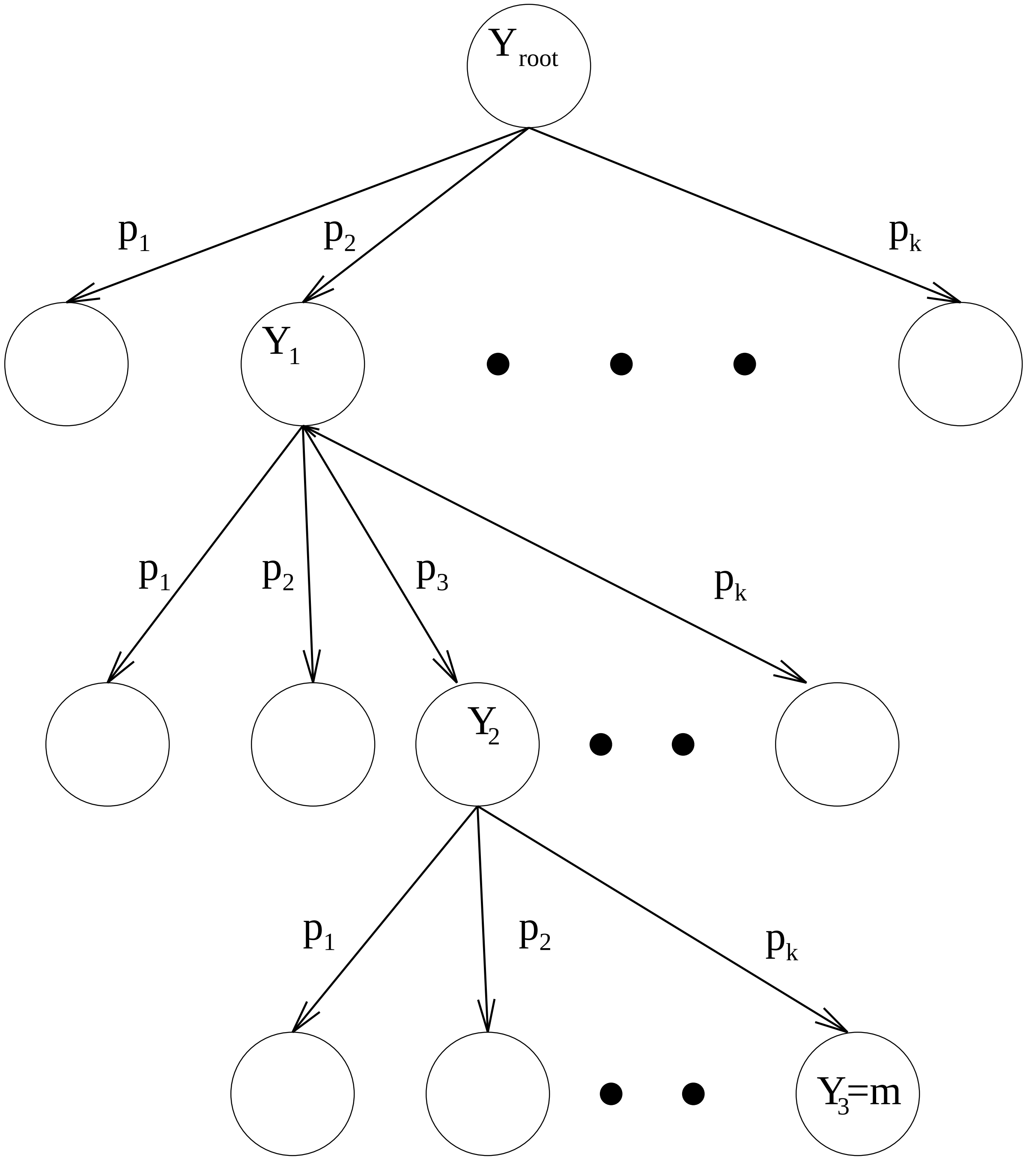

Figure 1: The signer's tree. Message m is signed using the path determined by P_2, P_3, P_k.

so that it is infeasible to factor ℓ-bit numbers (and so that the RSA assumption is assumed to hold for moduli of length ℓ). The outdegree of the tree is k, whereas (1) the public-key modulus, (2) the labels of the internal nodes, and (3) the elements of X are all ℓ bits long. We will also take the number of elements in X and size of the labels of the leaves to be ℓ, but as we discuss in Section 3.4, these may be smaller. The size of a signer's public key depends only on ℓ. As mentioned in Section 3.1, a possible choice for the parameters is taking ℓ and k to be about 1000, but there is no particular reason that they should be equal.

Shared Information: There are two sets $X = \{x_1, x_2, \ldots, x_\ell\}$ and $P = \{p_1, p_2, \ldots, p_k\}$ of integers, and a random ℓ-bit integer y_{root}. The x_i's are random integers of length at most ℓ. The p_i's are primes; they can be either the k smallest primes or k random primes of length ℓ, according to whether one relies on the general or stronger version of the RSA assumption (see exact formulation in Section 4). These lists are fixed and the same for all signers. They must be accessible to all signers and verifiers.

For ease of exposition we assume the depth d of the tree is fixed. However, this is not essential, and the signer is free to change d at any time.

The public key: the public key of each signer s is an ℓ-bit number N_s which is the product of two primes p_s and q_s. It is important to chose the primes at random (and independent of the list X) from all primes of length at most $\ell/2$. The prime factors p_s and q_s should be chosen so that for all $1 \le i \le k$ we have $p_i \not| (p_s - 1)$ and $p_i \not| (q_s - 1)$ and hence p_i is relatively prime to $\varphi(N_s)$. If P consists of large primes, then this is necessarily the case. If P consists of the list of small primes, then it is still possible to sample from the primes such that $p_i \not| (p_s - 1)$ and $p_i \not| (q_s - 1)$ for all $1 \le i \le k$. So the key generation procedure is efficient.

The secret key: the secret key of the signer is the factorization of N_s, *i.e.*, the pair (p_s, q_s). Given the factorization it is easy to compute $q_1, q_2, \ldots q_k$ where $q_i \equiv 1/p_i \bmod \varphi(N_s)$.

The basic authentication step: To authenticate the label y_v of a non-root vertex v, let z_v be the label of the parent of v and let v be the jth child of its parent. We use the bits of y_v as selectors of the elements of X. Let y_{vi} denote the ith bit of y_v, for $1 \le i \le \ell$. Then the basic authenticator $auth(y_v)$ is given by

$$\left(z_v \prod_{y_{vi}=1} x_i \right)^{q_j} \bmod N_s.$$

Verification of a basic authenticator: The authentication of y_v can be verified as follows. Given a string α purported to be $auth(y_v)$, and given also z_v, y_v, j, and the public lists P and X, compute

$$z_v \prod_{y_{vi}=1} x_i \bmod N_s$$

and check that it is equal to $\alpha^{p_j} \bmod N_s$.

Signature generation: Messages are associated with leaves in a depth-first, left-right manner; that is, when a new message m is to be signed, the message is assigned the leftmost leaf that has not yet been used; m serves as the label of the leaf. At all times the signer maintains a labeled path from the root to the leaf most recently accessed, together with the authenticators for the labels of the vertices on this path. All other previously used labels and authenticators may be erased, since they will not be used again. Assuming $J \geq 0$ messages have been signed so far, the algorithm for signing the $(J+1)$st message, say, m, is:

1. Assign to m the $(J+1)$st leftmost leaf w and label w with m.
2. For each node v along the path from *root* to w that has not yet been labeled, choose a random label y_v.
3. For each node v along the path from the root to w (excluding the root but including w), if y_v has not yet been authenticated, then assuming v is the jth child of its parent, compute the basic authenticator $auth(y_v)$ using p_j and the label of the parent of v.

A signature of a message m labeling a leaf w is therefore of the form

$$\langle m, (j_1, j_2, \ldots j_d), (y_1, y_2, \ldots y_{d-1}), (\alpha_1, \alpha_2, \ldots \alpha_d) \rangle$$

where

- $(j_1, j_2, \ldots j_d)$ is the description of the path from the root to w, that is, indices in the range $1, \ldots, k$ of the children.
- $(y_1, y_2, \ldots y_{d-1})$ are the labels of all the nodes along the path from the root to w, excluding the root, whose value is part of the public information, and w, whose value is m.
- $(\alpha_1, \alpha_2, \ldots \alpha_d)$ are the basic authenticators for all the nodes along the path, excluding the root but including w, i.e. $\alpha_i = auth(y_i)$, as computed in the basic authentication step described above.

Signature verification: Given a claimed signature of the form

$$\langle m, (j_1, j_2, \ldots j_d), (y_1, y_2, \ldots y_{d-1}), (\alpha_1, \alpha_2, \ldots \alpha_d) \rangle$$

it is verified by applying the basic authenticator verification procedure described above for all the labels of the nodes on the path, *i.e.*, $(y_1, y_2, \ldots y_{d-1}, m)$, and where the purported authenticators are $(\alpha_1, \alpha_2, \ldots \alpha_d)$. There is sufficient information to perform these verifications, since for each y_i the index of the node at its parent is given as j_i and the label of the parent is known as well – it is either in the list $(y_1, y_2, \ldots y_{d-1})$ or it is y_{root}.

3.3 Computational Requirements:

We now analyze the computational requirements of the various operations of the signature scheme. The computationally expensive operations that must be performed are modular exponentiation and subset product. If we try to compare the two in terms of the number of modular multiplications, then straightforward implementations require $\frac{3\ell}{2}$ and $\frac{\ell}{2}$ modular multiplications respectively (the $\ell/2$ figure is the expected one for *random* subsets). There are possible speed-ups for both operations, which we discuss in Section 3.4, however it seems that the subset multiplication is easier. Therefore we can (pessimistically) regard the complexity of the two operations as similar.

The Time Complexity of Signing

Signing a message involves

- d exponentiations mod N_s (*i.e.*, RSA computations).

- d subset multiplications, *i.e.*, multiplying a random subset of ℓ numbers. (The subsets are random for the internal nodes; for the leaves we can either assume that the number of 1's in the block is $\ell/2$ on the average, or instead Xor the message with a fixed and random string that is part of the public information.)

Assuming that $d = 3$ and that subset multiplication is roughly equivalent to modular exponentiation, we can say that the complexity of the scheme is at most six times that of RSA. Taking the relative complexity of subset multiplication and exponentiation to be 1 : 3 (as the $\frac{3\ell}{2}$ and $\frac{\ell}{2}$ figures indicates) we get a factor of only four.

The Time Complexity of Verification

Verifying a purported signature on a message involves

- d RSA verifications.

- d subset multiplications.

If the primes in P are large, then this corresponds to $2d$ times the complexity of plain RSA with a large public exponent. However, if we are using small primes, then the complexity of the subset multiplications dominates and we get that verification is significantly more expensive than verification in plain RSA with a small public exponent.

The Size of a Signature

A valid signature consists of $2d-1$ numbers, each ℓ bits long, plus d numbers of length $\log k$ to describe the path from the root to the leaf. Therefore under the assumption that $d = 3$ the size of a signature is roughly 5 times that of RSA. (See Section 3.4 for improvements.)

Storage Requirements and the Size of the Public Key

A public key is an ℓ-bit number (similar to RSA). Unlike RSA, the scheme requires the storage of the lists X and P, and of y_{root}, a total of $\ell^2 + (k+1)\cdot\ell$ bits. Apart from the lists X and P, the memory needed to run the signature scheme is not large: one need only maintain information along a single path from the root to a leaf, *i.e*, $d(\ell + \log k)$ bits.

3.4 Remarks on Implementation

For the case in which k and ℓ are both about 1000, roughly 2,000,000 bits (about 256K bytes) of common information must be accessible to every user. This is feasible if both signer and receiver are a "full" computer, but may be an obstacle in using the scheme in a smart card environment, as current generation smart cards have a memory capacity which is an order of magnitude smaller. However, as smart cards are becoming more powerful, storing 128K bytes in ROM on a smart card may not be impossible. Moreover, if, rather than choosing the primes p at random, we instead use the smallest 1000 primes (see Assumption 3), then 128K bytes should suffice.

A tempting possibility for cutting the memory requirements is to generate the shared random information in some pseudo-random manner and provide only a short seed. However, since the information is shared, all current techniques of cryptographic pseudo-randomness fail.

Assume $k = \ell = 1000$ and consider a particular path in the tree. If the tree is of height $d = 3$, then the path has 3 internal nodes: the root, and two others, say, v and w. Let y_{root}, y_v, and y_w, respectively, be the labels of these internal nodes. The first time a message associated with a child of w is signed, the signer must perform the computation needed to authenticate y_w using y_v; however, this information can be stored and used for the remaining $k-1$ (about 1000) messages associated with children of w. Similarly, the authentication of y_w using y_{root} can be re-used k^2 times (about $1,000,000$ times). Therefore, for this reasonable choice of parameters, the *amortized* cost of signing a message in our scheme is at most twice that of signing using plain RSA.

We have ignored the issue of when the q_i's (the inverses $\bmod\ \varphi(N_i)$ of the p_i's) should be computed. To speed the signing, the signer can pre-compute them. However, this requires storing them in a secure memory. An alternative is to compute them on the fly. This does not add significantly to the computational load, since computing inverses is much easier than exponentiation. From the discussion above, if the signer stores the q_i's corresponding to the current path ($d-1$ values), then it has to compute at most one q_i per signature.

As specified in Assumption 3 below, the p_i's may be chosen to be small in order to expedite verification of a signature. Furthermore, Fiat [15] suggested a way of amortizing RSA computations. His method fits very well with our scheme, particularly since we use different roots. Fiat's method may be used to decrease the *size* of the signature as well: instead of storing for each y_v along the path the value $auth(y_v)$ separately, it is sufficient to provide one value in $Z^*_{N_s}$ from which all the $auth(y_v)$'s along a path are extractable. Therefore the size of a signature is only d numbers, each ℓ bits long, plus the description of the path. The only additional restriction is that all the p_i's of a path should be different. This does not reduce the total number of paths by much, instead of k^d paths we have $k \cdot (k-1) \cdots (k-d+1)$. An alternate way of reducing the signature size is to eliminate $(y_1, y_2, \ldots, y_{d-1})$. One can reconstruct their presumed values from $y_d = m$ and $(\alpha_1, \alpha_2, \ldots \alpha_d)$ and check it using y_{root}: starting from $h = d$ down to $h = 1$ compute

$$y_{h-1} = \frac{\alpha_h^{p_j}}{\prod_{y_{hi}=1} x_i} \bmod N_s$$

and verify that $y_0 = y_{root}$.

We have fixed the size of X to be ℓ which is also the size of the labels of the nodes in the tree and the number of bits in the public key. In principal we could use fewer x_i's, which would imply a shorter public list X and multiplication of smaller subsets. However, this would require us in the basic authentication step to hash y_v and to use the bits of $h(y_v)$ in the selection of the elements of X. It is sufficient to apply a *universal one-way hash function* [25] (see definition and discussion in Section 5), but in order to obtain existential unforgeability using *only* the RSA Assumption (of either type), rather than relying on the properties of the hash functions, we did not present it as such.

Other known techniques for speeding up RSA and the Fiat-Shamir signature scheme are applicable to our scheme as well. The signer can perform its calculations modulo each of the factors of N_s separately, and then combine them using Chinese remaindering. This is true for both the exponentiation and the subset multiplication. Therefore, for this choice of parameters, the performance comparison with plain RSA is valid even if one assumes that the implementation of RSA does these (quite common) optimizations. To expedite the subset product computations, one can preprocess the list X, partitioning it to small sets, say pairs or triples, and for every set computing all products of its subsets. This decreases the time spent on subset multiplication at the cost of additional storage. For instance, if one partitions X into triples and preprocesses them, then storing the result requires 7/3 times the space required to store X. The expected number of multiplications to compute a random subset decreases from $\ell/2$ multiplications (without preprocessing) to $\frac{7}{8} \cdot \frac{\ell}{3} = \frac{7\ell}{24}$ multiplications (with the preprocessing).

4 Security of the Scheme

In this section we prove that our signature scheme is existentially unforgeable under adaptive plaintext attack [20] (see section 2). Specifically, we show that the ability to generate a single $(m, S(m))$ pair, for any m on which the signer was not explicitly asked to sign – even for a nonsensical m, – violates the RSA Assumption(s) specified below:

Assumption 2 *For every polynomial $q(x)$ there exists an ℓ_0 such that for all $\ell \geq \ell_0$ the following holds. Let N be a number of length at most ℓ bits which is chosen as the product of two large random primes of equal length, let p be a random prime of length ℓ bits and let m be a random ℓ-bit number. For every probabilistic $q(\ell)$ time bounded algorithm $\mathcal{F}$:*

$$\Pr[\mathcal{F}(N, p, m^p \bmod N) = m] < \frac{1}{q(\ell)}$$

where the probability is over the choices of N, p, m, and the internal coin flips of $\mathcal{F}$.

Assumption 3 *For every polynomial $q(x)$ there exists an ℓ_0 such that for all $\ell \geq \ell_0$ the following holds. Let p be* any *prime. Let N be a number of length at most ℓ bits which is chosen as the product of two large random primes of equal length and such that p is relatively prime to $\phi(N)$, and let m be a random ℓ-bit number. For every probabilistic $q(\ell)$ time bounded algorithm $\mathcal{F}$:*

$$\Pr[\mathcal{F}(N, p, m^p \bmod N) = m] < \frac{1}{q(\ell)}$$

where the probability is over the choices of N and m and the internal coin flips of $\mathcal{F}$.

Intuitively, the security of the scheme rests on the important observations made in [34] and [16] respectively:

- Having a black box that computes $x^{1/p_1} \bmod N$ for random x does not help in evaluating $x^{1/p_2} \bmod N$, if p_1 and p_2 are relatively prime.
- For numbers $x_1, x_2, \ldots x_\ell$, for arbitrary subset $S \subset \{1, \ldots, \ell\}$ and random $y \in Z_N^*$, the value of $(y \prod_{i \in S} x_i)^{1/p} \bmod N$ yields no information about any of the $x_i^{1/p} \bmod N$.

Suppose that the scheme can be broken, *i.e.,* there is an algorithm $\mathcal{A}$ that operates in time T and has probability ρ of breaking the scheme. We show that there is an algorithm $\mathcal{B}$ that works in expected time $\tilde{O}(T)$ and that can extract pth roots with probability at least $\frac{\rho}{\ell \cdot k}$ where $\tilde{O}$ hides factors that are polynomial in ℓ and k but independent of T. (See a remark on refining the analysis following Theorem 4.1)

The input to $\mathcal{B}$ is the triple (x, N, p), where x, N and p are as in Assumption 2. (The argument for Assumption 3 can be treated similarly.) The desired output is $x^{1/p}$ mod N. Algorithm $\mathcal{B}$ consists of three phases, a preprocessing phase, in which the public key and public information are generated, and a simulation phase, in which the algorithm $\mathcal{A}$ is simulated on the public key generated in the previous phase. If the simulated $\mathcal{A}$ is successful in breaking the signature scheme then a third phase takes place, in which $\mathcal{B}$ extracts $x^{1/p}$ mod N from the illegitimate signature produced by $\mathcal{A}$.

If a tree-based scheme (*i.e.*, a system where parents vouch for the authenticity of their children) is broken, then there must be the first time an illegitimate value (*i.e.* a value not authenticated by the signer) is authenticated by some node w. We can guess with probability $1/k$ at which child, $1 \leq j \leq k$, of w this will occur (note that there is no need to guess w itself). Furthermore, with probability at least $1/\ell$ we can guess an index $1 \leq i \leq \ell$ at which the legitimate value authenticated with w and p_j differs from the forged value. $\mathcal{B}$ attempts to simulate $\mathcal{A}$ and make the above guesses. If the simulated $\mathcal{A}$ is successful in breaking the signature scheme and if the guesses turn out to be correct, then from this information it should possible to extract $x^{1/p}$ mod N.

4.1 Detailed description of algorithm $\mathcal{B}$

The input to $\mathcal{B}$ is the triple (x, N, p) and the desired output is $x^{1/p}$ mod N. In the preprocessing phase $\mathcal{B}$ creates a labeled tree and the lists X and P. These are then used in the simulation phase to enable the interaction with (*i.e.*, signing at the request of) the simulated $\mathcal{A}$. If the simulated $\mathcal{A}$ is successful in breaking the signature scheme, then there is another (short) phase, during which $\mathcal{B}$ extracts the desired value $x^{1/p}$ mod N from the forged signature produced by the simulated $\mathcal{A}$. Let $[K]$ denote the set $\{1, \ldots, k\}$.

Algorithm $\mathcal{B}$: Preprocessing Phase

1. Choose uniformly at random random $1 \leq i \leq \ell$ and $1 \leq j \leq k$.

2. Set $N_s = N$.

3. Choose random ℓ-bit primes $p_1, p_2, \ldots p_{j-1}, p_{j+1}, \ldots p_k$. Set $p_j = p$ and let $P = \{p_1, p_2, \ldots, p_k\}$.

4. To generate $x_1, x_2, \ldots x_{i-1}, x_{i+1}, \ldots x_\ell$, select uniformly at random $\ell - 1$ values in $Z^*_{N_s}$ and raise each one of them to the power $p_1 \cdot p_2 \cdots p_k$ modulo N_s. Thus for all $h \in [K] \setminus \{j\}$, all the p_hth roots of the elements in $X \setminus \{x_i\}$ are known and can be computed efficiently. To generate x_i, select s uniformly at random from $Z^*_{N_s}$, let r be s^{p_j} mod N_s and set $x_i = (r \cdot x)^{p_1 \cdots p_{j-1} \cdot p_{j+1} \cdots p_k}$ mod N_s. Note that

   - For each $h \in [K] \setminus \{j\}$, x_i^{1/p_h} mod N_s can be computed efficiently.
   - Given x_i^{1/p_j} mod N_s we can extract $(r \cdot x)^{1/p_j}$ mod N_s, and hence x^{1/p_j} mod N_s, efficiently as follows. Let a and b be such that $a \cdot p_j + b \cdot p_1 \cdots p_{j-1} \cdot p_{j+1} \cdots p_k = 1$ (they exist and are easy to find by the GCD algorithm). Then

$$(r \cdot x)^{1/p_j} = \left((r \cdot x)^{p_1 \cdots p_{j-1} \cdot p_{j+1} \cdots p_k / p_j} \right)^b \cdot (r \cdot x)^a \bmod N_s$$

so

$$x^{1/p_j} = (x_i^{1/p_j})^b \cdot r^a \cdot x^a / s \bmod N_s.$$

5. Recall that T is the upper bound on the running time of $\mathcal{A}$ and hence an upper bound on the number of signatures requested by the simulated $\mathcal{A}$. We now determine y_v for all internal nodes v that are ancestors of the first T leaves. The goal is to choose the labels so as to be able to provide any signatures requested by the simulated $\mathcal{A}$. The tree is constructed in a bottom up fashion. Recall that i and j are fixed by now, and that by the construction of x_i we cannot efficiently compute $x_i^{1/p_j} \bmod N_s$. Thus, we want to ensure that if v is the jth child of its parent, then when authenticating y_v it is not necessary to compute the p_jth root of x_i. We can ensure this as follows. Letting y_{vi} denote the ith bit of y_v,

   - if $y_{vi} = 1$ then z_v (the label of the parent of v) should be

     $$\beta_v^{p_1 \cdot p_2 \cdots p_k} / x_i \bmod N_s$$

     for random $\beta_v \in Z^*_{N_s}$.
   - if $y_{vi} = 0$, then z_v should be $\beta_v^{p_1 \cdot p_2 \cdots p_k} \bmod N_s$ for random $\beta_v \in Z^*_{N_s}$.

   This suggests a specific bottom-up method for choosing the y_v's. However, for technical reasons we prefer the following two-step description:

   (a) For each node v in the subtree containing the first T leaves, choose a random bit b_v and set $y_{vi} = b_v$, where y_{vi} denotes the ith bit of y_v. (For a leaf v, b_v is necessarily a guess, and is correct with probability 1/2. However, all the other b_v's will indeed be the values of y_{vi}.)

   (b) For all internal vertices v, let u be the jth child of v and let $b_v = y_{vi}$ and $b_u = y_{ui}$ be the values chosen in Step 5a. Choose a random $\beta_v \in Z^*_{N_s}$. If $y_{ui} = 1$, then compute the candidate value

   $$y_v = \beta_v^{p_1 \cdot p_2 \cdots p_k} / x_i \bmod N_s.$$

   If $y_{ui} = 0$, then compute the candidate value

   $$y_v = \beta_v^{p_1 \cdot p_2 \cdots p_k} \bmod N_s.$$

   Note that in both cases y_v is uniformly distributed in $Z^*_{N_s}$. With probability 1/2 we have that $y_{vi} = b_v$ (*i.e.*, the ith bit of y_v agrees with the value chosen for it in Step 5a). If they are equal, then keep the candidate y_v and continue with the preprocessing; otherwise, repeat this step, choosing new candidate values for y_v (by choosing new random $\beta_v \in Z^*_{N_s}$) until successful.

   Let y_{root} denote the value assigned to the root by the above procedure.

Before we describe the simulation we should note that in the description above there is one inaccuracy: we choose values $z \in Z^*_{N_s}$ at random (where z stands for either the x_k's or the y_v's), whereas in a regular execution of $\mathcal{A}$ it should be that z is a random ℓ-bit number.

This can be corrected by replacing z by a random z' of length ℓ such that $z \equiv z' \bmod N$. We should not worry about the numbers that are not relatively prime to N_s, since they are rare. However, if $z < (2^\ell \bmod N)$ then there are $\lceil 2^\ell/N \rceil$ ℓ-bit values z' such that $z \equiv z' \bmod N$ and if $z \geq (2^\ell \bmod N)$ then there are $\lfloor 2^\ell/N \rfloor$ ℓ-bit values z' such that $z \equiv z' \bmod N$. This gives a certain advantage to the latter. To correct this bias, after z is selected uniformly from $Z^*_{N_s}$ we reject a $z \geq (2^\ell \bmod N)$ (and repeat the process in which they were chosen) with probability $\frac{\lfloor 2^\ell/N \rfloor}{\lceil 2^\ell/N \rceil}$. This cannot increase the expected work by more than a factor of 2.

Algorithm $\mathcal{B}$: Simulation Phase

1. Invoke $\mathcal{A}$ with (X, P, N_s, y_{root}) as determined in the preprocessing phase.
2. Start the simulation of $\mathcal{A}$; at every step t, $\mathcal{A}$ queries the signer with a message m_t and requests a signature. The algorithm $\mathcal{A}$ receives as requested a signature on m_t using the path to the tth leftmost leaf of the tree chosen in the preprocessing. The signature is generated according to the following:
   - To handle a node which is not the jth child of its parent is easy, since we can extract all p_hth roots when $h \neq j$.
   - To handle a node which is a jth child but is an *internal* node can also be done easily, due to the way y_v was chosen in step 5 in the preprocessing phase.
   - When handling a leaf u that is the jth child of its parent v, then with probability $1/2$ the incorrect b_u was chosen, that is, for which b_u does not equal the ith bit of the message m_t for which $\mathcal{A}$ requests a signature. (Note that $\mathcal{A}$ does not receive the guesses b_u of the leaves and hence its choice of messages is independent of them.) In this case
     - Rewind $\mathcal{A}$ for j steps, back to the stage just before the parent of v is used for the first time.
     - Choose at random a new value for b_u.
     - Choose a new value y_v, where v is the parent of u; the value of y_v should satisfy the constraint $y_{vi} = b_v$ imposed in the preprocessing step, thus preventing further propagation of the rollback. This is done as in Step 5b of the preprocessing.

   As discussed below, the rewinding may increase the *expected* run-time by a factor of at most two.

Algorithm $\mathcal{B}$: The Extraction Phase

Suppose that $\mathcal{A}$ is successful and produces a signature

$$\langle m, (j_1, j_2, \ldots j_d), (y_1, y_2, \ldots y_{d-1}), (\alpha_1, \alpha_2, \ldots \alpha_d) \rangle$$

where the signature passes the verification test, but m was not signed in the simulation phase. Among $(y_0 = y_{root}, y_1, y_2, \ldots y_{d-1}, y_d = m)$ there must be the first (smallest) $1 \leq a \leq d$ such that, letting v be the node reached by following the path $j_1, j_2, \ldots, j_a$, y_a is different from the label v was assigned in the Preprocessing Phase. Let y_v be the true label of v (assigned in the Preprocessing Phase) and let

$$\alpha \doteq auth(y_v) = \left(y_{a-1} \prod_{y_{vh}=1} x_h \right)^{q_{j_a}} \bmod N_s.$$

By Step 5 of the Preprocessing phase, if v is a leaf then α can be computed by $\mathcal{B}$; if v is an internal node then α was computed by $\mathcal{B}$. The extraction is successful if the following two conditions hold: $j_a = j$ and $y_{vi} \neq y_{ai}$ (that is, y_a and y_v differ in the ith bit). If this is the case, then from the forged signature we have $\alpha_i = \left(y_{a-1} \prod_{y_{ah}=1} x_h \right)^{q_j} \bmod N_s$, and from above we have $\alpha = \left(y_{a-1} \prod_{y_{vh}=1} x_i \right)^{q_j} \bmod N_s$. We can now obtain $x_i^{q_j} \bmod N_S$ as follows. Assume without loss of generality that $y_{vi} = 1$. Let $S_1 = \{h \neq i | y_{ah} = 1 \ \wedge \ y_{vh} = 0\}$ and $S_2 = \{h \neq i | y_{ah} = 0 \ \wedge \ y_{vh} = 1\}$. Then we have

$$\frac{\alpha}{\alpha_i} \cdot \frac{\prod_{h \in S_1} x_h^{q_j}}{\prod_{h \in S_2} x_h^{q_j}} = \frac{(y_{a-1} \prod_{y_{vh}=1} x_i)^{q_j}}{(y_{a-1} \prod_{y_{ah}=1} x_h)^{q_j}} \cdot \frac{\prod_{h \in S_1} x_h^{q_j}}{\prod_{h \in S_2} x_h^{q_j}} = x_i^{q_j} \bmod N_s.$$

As discussed in Step 4 of the Preprocessing phase, for all $h \neq i$ it is easy to compute $x_h^{q_j} \bmod N_s$ and from $x_i^{q_j} \bmod N_S$ it is easy to extract $x^{q_j} = x^{1/p} \bmod N$, the desired output.

4.2 Analysis of Algorithm $\mathcal{B}$

When $\mathcal{A}$ engages in an interaction with a signing process, there is a distribution on the transmissions that $\mathcal{A}$ witnesses. The proof of security rests on the fact that this distribution and the one $\mathcal{A}$ witnesses in the simulation are similar. Let $\mathcal{D}(\mathcal{A})$ be the distribution $\mathcal{A}$ witnesses in a regular execution, conditioned on the event that every $x \in X$ and all the labels of the internal nodes are relatively prime to N_s (the probability that this is not the case is bounded by $T \cdot \frac{N - \varphi(N)}{N} \leq \frac{3T}{\sqrt{N}}$ which we assume is negligible). This distribution is simple to describe: it consists of a public key N_s distributed as in Assumption 2, a list X of ℓ numbers uniformly distributed over $0 \dots 2^\ell - 1$ and relatively prime to N_s, a list of random primes of length ℓ, and for each internal node which is an ancestor of one of the first T leaves, a random value which is uniformly distributed over $0 \dots 2^\ell - 1$ and relatively prime to N_s. All the other values are determined by these values and the messages on which $\mathcal{A}$ requests a signature.

Let $\mathcal{D}(\mathcal{B})$ denote the distribution that $\mathcal{A}$ witnesses in the simulated interaction with $\mathcal{B}$.

Claim 4.1 *The distributions $\mathcal{D}(\mathcal{A})$ and $\mathcal{D}(\mathcal{B})$ are identical. Furthermore, the distribution $\mathcal{D}(\mathcal{B})$ is independent of the choice of i and j in Step 1 of the Preprocessing Phase.*

Proof. The distribution $\mathcal{D}(\mathcal{B})$ consists of the lists X and P, the public key N_s, and the labels of the internal nodes that are ancestors of the first T leaves. All other information is determined by these and the messages that $\mathcal{A}$ provides. The list X consists of ℓ numbers distributed uniformly and independently among the ℓ-bit numbers relatively prime to N_s. This follows from the way they are chosen in Step 4 of the Preprocessing Phase (note that the choice of r randomizes x in $Z^*_{N_s}$). Therefore X is also independent of the choice of i.

The list P is a list of k random primes of length ℓ, given that the input $p = p_j$ was chosen at random (which is our assumption). Again, it is independent of j. The situation for the labels of the internal nodes may seem more delicate, since these values are chosen by a more complex process. Consider the nodes v that are not parents of leaves. Their labels are chosen at Step 5 of the Preprocessing Phase and never change thereafter. Recall that this is done by first picking the ith bit as a random $b_v \in \{0,1\}$ and then choosing the rest of the label by either computing $\beta_v^{p_1 \cdot p_2 \cdots p_k}/x_i \bmod N_s$ or $\beta_v^{p_1 \cdot p_2 \cdots p_k} \bmod N_s$ (depending on the ith bit of the label of the jth child of v) for a random $\beta_v \in Z^*_{N_s}$ until the ith bit of the result equals b_v. From the randomness of the bit b_v and of the value β_v, the label y_v is uniformly distributed.

The labels of the nodes v that are parents of leaves may be re-chosen at Step 2 of the Simulation Phase. Note that for a node v and its jth child u the values for y_v and b_u chosen at Step 5 of the Preprocessing are independent. Therefore the event of choosing a new label for the node v is independent of the value chosen for y_v. (It happens when the guess b_u of the ith bit of the message signed by the jth child of v was wrong; recall that $\mathcal{A}$ does not receive information regarding b_u.) Therefore the new value for y_v is again uniformly distributed and independent of the new b_u. $\square$

Claim 4.2 *The expected time to run the simulation is $\tilde{O}(T)$.*

Proof. There are two possibilities that may force us to spend more than $\tilde{O}(T)$ steps (recall that the $\tilde{O}$ hides factors that are polynomial in ℓ and k but independent of T): one is Step 5 of the preprocessing phase, where we may fail to choose y_v that satisfies the requirement on y_{vi}. However this happens with probability 1/2 and hence at most doubles the expected amount of work. The second possibility is in Step 2 of the simulation, where we may have to rewind $\mathcal{A}$ for j steps. However, as before, this happens with probability 1/2 and increases the expected work by a factor of $\frac{k+j}{k} \leq 2$ (since the first j children of a node which is a parent of leaves may be repeated with probability 1/2 and the rest are developed once). $\square$

Claim 4.3 *The probability of success is at least $\frac{\rho-(3T/\sqrt{N})}{k \cdot \ell}$.*

Proof. The probability that $\mathcal{A}$ breaks the system in a regular execution, given that all the x_i's and y_v's are relatively prime to N, is at least $\rho - 3T/\sqrt{N}$. Since $\mathcal{D}(\mathcal{A})$ and $\mathcal{D}(\mathcal{B})$ are identical and the latter is is independent of i and j chosen by $\mathcal{B}$, the probability $\mathcal{A}$ in the simulation breaks the scheme is at least $\rho - 3T/\sqrt{N}$.

Suppose that $\mathcal{A}$ breaks the signature scheme and produces a forged signature,

$$\langle m, (j_1, j_2, \ldots j_d), (y_1, y_2, \ldots y_{d-1}), (\alpha_1, \alpha_2, \ldots \alpha_d) \rangle$$

and suppose that a, $1 \leq a \leq d$, is the least a such that y_a is different from the true label of the corresponding node v. Let y_v be the true label assigned to v in the Preprocessing Phase of $\mathcal{B}$. Recall from the Extraction Phase of $\mathcal{B}$ the condition for a successful extraction of $x^{1/p} \bmod N$: $j_a = j$ and $y_{ai} \neq y_{vi}$. However, j and i are independent of $\mathcal{D}(\mathcal{B})$ and hence of j_a, y_a and y_{vi}. Therefore given such a forged signature $\Pr[j_a = j] = \frac{1}{k}$ and $\Pr[y_{ai} \neq y_{vi}] \geq \frac{1}{\ell}$. The two events ($j_a = j$ and $y_{ai} \neq y_{vi}$) are independent, since the choice of i and j is independent. Therefore with probability at least $\frac{1}{k \cdot \ell}$ they both occur and, as explained in the Extraction Phase of $\mathcal{B}$, we can extract $x^{1/p} \bmod N$. $\square$

We can therefore conclude that any algorithm for breaking the provided signature scheme can be used at a related cost and probability of success to extract modular roots:

Theorem 4.1 *Any algorithm $\mathcal{A}$ that breaks the scheme in time T with probability ρ can be turned into an algorithm for breaking Assumption 2 in time $\tilde{O}(T)$ and success probability $\frac{\rho-(3T/\sqrt{N})}{k\cdot\ell}$.*

We remark that it is possible to modify our scheme so that any algorithm for breaking the scheme operating in time $\tilde{O}(T)$ and probability of success ρ can be converted into an algorithm for breaking the RSA assumption in time T and probability of success $\Omega(\frac{\rho}{k})$ (rather than $\frac{\rho}{k\cdot\ell}$ as above). The modification is based on an idea of Even, Goldreich and Micali [14]. Each value to be authenticated using *auth* is first encoded with a code C that has a large relative distance, *i.e.*, the Hamming distance of any two codewords is a large fraction, γ, of their length. The length of the list X should now be ℓ', the length of codewords of C, rather than ℓ. For a good code $\frac{\ell'}{\ell}$ should be $O(1)$. The basic authentication step is done by

$$\left(z_v \prod_{C(y_v)_i=1} x_i\right)^{q_j} \bmod N_s.$$

Now, at the beginning of the preprocessing phase, when we guess an $1 \leq i \leq \ell'$, the probability that the authenticated value and the forged value differ at the ith bit is at least γ, since they are encoded using C. Therefore the overall probability of success is $\gamma \cdot \rho/k$.

A finer analysis of the running time of $\mathcal{B}$ can be done by separating the running time of $\mathcal{A}$ into T_1, the number of (message, signature) pairs obtained by $\mathcal{A}$, and T_2, its internal running time. The running time of $\mathcal{B}$ is $\tilde{O}(T_1) + O(T_2)$.

5 Application to Signing Faxed Documents

In this section we describe how to apply our signature scheme, or any existentially unforgeable scheme, to signing faxed documents. This application can be seen as an instance of a document repository, mentioned in the Introduction. In particular, we describe how to use such a scheme in order to obtain short receipts for long documents.

As pointed out in the Introduction, in designing a signature scheme appropriate for fax documents it is important to distinguish between the printed document, denoted P, and the bit stream image of the scanned document, denoted D, that results in the printing of P. In general, once P is printed from D, it is impossible to reproduce D precisely: re-scanning the printed document P will likely yield a different bit stream $D' \neq D$. Therefore, for digital signatures of faxed documents to be useful, the semantics and responsibilities of each of the sides (sender and receiver) should be determined carefully.

If neither the sender nor the receiver has a long-term storage device, then providing meaningful digital signatures, verifiable by a third party, seems to be an impossible task. We therefore assume that at least one of the sender or the receiver is using a computer-FAX, or *CFAX*, system, *i.e.*, a machine that can permanently store the scanned document. The simple FAX machine is not equipped with long term storage, but it is equipped with a processor that can perform any required computation on the stream of data passing through the FAX. There are three scenarios to consider:

- CFAX to CFAX - straightforward, *i.e.* not different from any computer communication, the receiving CFAX simply stores the bits representing the image of the signed document together with the signature.
- CFAX to FAX - interesting (see below).
- FAX to CFAX - same as CFAX to CFAX.

Assume we have a CFAX machine, denoted C, sending a document to an ordinary FAX, denoted F. This scenario is applicable whenever we have a large organization communicating with many independent agents or clients, *e.g.* a bank with its clients or an insurance company with its agents. Although the receiver F is not assumed to be able to store the bit-stream image D of the faxed document, it is this bit-stream, and *not* the printed version of the document (which we can assume is available), that has been digitally signed. Thus, the sender C must store the data. Since the receiver may not trust C to produce the data on demand, C should issue to the receiver some type of receipt. Therefore, in this scenario the receiver cannot alone convince a third party that the document was signed by the claimed sender, but it can show to a third party a signed statement promising cooperation in the judgment. As pointed out below, refusal on the part of the sender to cooperate may actually permit the receiver to forge, so it is definitely in the sender's interest to cooperate. In the following, we do not assume that the receiver has a long-term storage device.
Problem formulation: The problem raised by the discussion above can be formulated as follows: design a signature scheme that allows a signer to provide to a receiver F *short* receipts $R(D)$ on *long* documents D. Given $R(D)$ and D it should be easy for a third party to determine whether $R(D)$ is indeed a signature on D (this is the usual requirement from a signature scheme). The CFAX to FAX scenario imposes two additional requirements:

- Given $R(D)$ *alone*, it should be easy to determine that it is legitimate – it should be computationally infeasible for the receiver F to produce *any* legitimate looking receipt $R(D)$, except for those provided by C.
- Given $R(D)$, the signer C should be able to produce only one corresponding D.

To apply a solution to this problem in the context of signing faxes, one should print the the document D, as is customary with faxes, and also print either in hex or in some more area-efficient method, such as a two-dimensional bar code, the receipt $R(D)$.
Cryptographic tools: Since the requirement is that the scheme produce short signatures on long documents, it is quite clear that some sort of one-way hashing should be used. These come in (at least) two flavors: *universal one-way hash functions* [25] and *collision intractable functions* [10]. A family H of *universal one-way hash functions* has the following property: *Fix a string S. Let h be chosen at random from the family H of universal one-way hash functions. Then it is computationally infeasible to find a string S' such that $h(S) = h(S')$.* This is weaker than collision-intractability, which allows S to be chosen *after* h is known, *i.e.* given a random h it should be infeasible to find different S and S' such that $h(S) = h(S')$. However, it may be easier to construct, or at least to prove secure, universal one-way hash functions, and constructions are known to exist under general assumptions [25, 31] (for more reasons why it may be preferable to assume only the existence of universal one-way hash

functions, see [5]). In our context, the strings S are just documents. Thus for any fixed document D, if h is chosen at random from the family H of universal one-way hash functions then it is computationally infeasible to find a document D' such that $h(D) = h(D')$.

One concrete proposal for constructing universal one-way hash functions, due to Impagliazzo and Naor [21], is based on the subset sum problem (they also propose some less efficient schemes based on factoring). In particular, breaking the assumed universal one-way hash property of this family is proved in [21] to be as hard as solving a random subset sum problem. Recently Ajtai [2] showed that breaking such functions implies the ability to solve several worst-case lattice problems. On the other hand, Schnorr and Hörner [33] (and ref. therein) provide computational experience in solving such problems, which implies bounds on the choice of parameters.

Collision-intractable hash functions can be constructed based on the discrete logarithm problem [8, 9]. Alternatively, as was pointed out by [19], Ajtai's results imply that the Impagliazzo-Naor construction is actually collision-intractable (assuming the above-mentioned worst-case lattice problems are hard).

There are also various proposals for fast one-way hash functions, like MD5 [29] and SHA [1], whose security is not treated formally. This does not mean that they are useless, but the goal of this paper is to provide a solution that is efficient *and* provably secure. Note that collisions have been found in MD5 [12].

A nice property for the one-way hash function to have is that it is easy to compute it on the fly, without storing D. All the above proposals enjoy this property, or can be easily adapted to have it.

We should also assume the existence of an existentially unforgeable signature scheme secure against chosen plaintext attack, such as the one described in Section 3.

Key Management: There are standard ways to avoid having to maintain and access a directory of public keys. For example, there can be a central agency with which public keys are registered. The central agency has its own pair of keys, K_{center}, L_{center}, where all users know K_{center} (rather than having to know all public keys).

The Scheme: In the following, $\langle a, b\rangle$ denotes the concatenation of a and b. The family H of universal one-way hash functions should be chosen so that the number of bits in the pair $\langle h(D), h\rangle$ for any $h \in H$ and any document D, can be signed with a single application of the signature function. The CFAX sender, C, first forwards to the recipient the statement "C's public signature key is K_C," signed with the center's signature key. The recipient, F, knows K_{center} and can therefore be certain of using the correct public key for C. Let $S_C(m)$ denote a signature on message m with C's key. The agents proceed as follows.

1. F chooses at random an $h \in H$. F *does not* reveal h to C.

2. C sends to F the document D; F hashes D on-line, computing $h(D)$ and temporarily saving this; it also prints D (on-line).

3. F sends h to C.

4. C computes $h(D)$ and sends to F $S_C(\langle h(D), h\rangle)$, that is, C's the signature on the concatenation of $h(D)$ and h.

5. Let α be the message received by F. Then F verifies that α is indeed $S_C(\langle h(D), h\rangle)$ using h and $h(D)$ computed and stored above. F then prints $\langle h(D), h\rangle, S_C(\langle h(D), h\rangle)$ in hex or using a two dimensional bar code (a more compact and robust representation). This printout should be kept in a safe place, since it is the recipient's only proof of the authenticity of the document.

For particularly important transactions F may store D on tape, or even print the bit-stream of D itself. This is discussed next.

Handling Disputes: The tuple $(\langle h(D), h\rangle, S_C(\langle h(D), h\rangle)$ constitutes a promise by C to produce a document D' such that $h(D') = h(D)$. Whenever the need arises, say, that a third party wants to check the receiver's claim that he received a specific printed document form the sender, C must produce D'. (We assume that given D' it is possible to verify that it corresponds to the printed document.) We claim that in the above protocol both parties are protected:

- Protection of F: Since D was fixed by C without knowledge of h, if $D' \neq D$, then this means that C broke the universal one-way hash function, because it should be intractable to find a D' such that $h(D') = h(D)$.
- Protection of C: Since the signature scheme is existentially unforgeable, F cannot produce any tuple $(\langle h(D), h\rangle, S_C(\langle h(D), h\rangle)$ that was not originally produced by C.

If the sender C refuses to produce an appropriate D, then this can be treated as a breach of contract. (or "evidence" that C indeed signed a document corresponding to the printed one). Up to this point we have assumed that F has no long-term storage medium. Suppose instead that F records documents on tape, or some other low-cost (but not easily searchable) medium. In this case, if C does not cooperate, then F could produce the tape. Moreover, this last option may have a "nasty" surprise for C: since F is the one who chose h, C's refusal to cooperate exposes C to "forgery" of the receipt: if F had been dishonest and chosen h dependent on D (after Step 2, rather than before Step 2), then F may be able to produce $D' \neq D$ such that $h(D') = h(D)$.

Acknowledgements

We thank the two diligent and anonymous referees and Oded Goldreich, whose many comments helped improve the paper.

References

[1] FIPS 180-1, *Secure Hash Standard*, NIST, US Dept. of Commerce, Washington D. C. April 1995.

[2] M. Ajtai, *Generating Hard Instances of Lattice Problems*, Proc. 28th ACM Annual Symposium on the Theory of Computing, 1996, pp. 99–108.

[3] M. Bellare and S. Micali, *How to Sign Given Any Trapdoor Function*, J. of the ACM **39**, 1992, pp. 214–233.

[4] M. Bellare and P. Rogaway, *The exact security of digital signatures: How to sign with RSA and Rabin*, Advances in Cryptology - Eurocrypt '96, Lecture Notes in Computer Science vol. 1070, Springer-Verlag, 1996.

[5] M. Bellare and P. Rogaway, *Collision-Resistant Hashing: Towards Making UOWHFs Practical*, Advances in Cryptology - Crypto '97, Lecture Notes in Computer Science, Springer-Verlag, 1997.

[6] D. Bleichenbacher, *Generating El Gamal signatures without knowing the secret key*, Advances in Cryptology - Eurocrypt '96, Lecture Notes in Computer Science, Vol. 1070, Springer Verlag, 1996.

[7] J. Bos and D. Chaum, *Provably Unforgeable Signatures*, Advances in Cryptology - Crypto'92, Lecture Notes in Computer Science vol. 740, Springer Verlag, 1993, pp. 1–14.

[8] S. Brands, *An efficient off-line electronic cash system based on the representation problem*, CWI Technical Report, CS-R9323, 1993.

[9] D. Chaum, E. van Heijst and B. Pfitzmann, *Cryptographically strong undeniable signatures, unconditionally secure for the signer*, Advances in Cryptology - Crypto '91, Lecture Notes in Computer Science 576, Springer Verlag, 1992, pp. 470–484.

[10] I. B. Damgard, *Collision Free Hash Functions and Public Key Signature Schemes* , Advances in Cryptology - Eurocrypt'87, Lecture Notes in Computer Science vol. 304, Springer Verlag, 1988, pp. 203–216.

[11] W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Trans. on Information Theory 22(6), 1976, pp. 644–654.

[12] H. Dobbertin, *Cryptanalysis of MD5 Compress*, presented at Eurocrypt'96 rump session.

[13] T. El Gamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. Inform. Theory, IT-31(4), 1985, pp. 469–472

[14] S. Even, O. Goldreich, and S. Micali, *On-line/Off-line Digital Signatures*, Journal of Cryptology, Vol 9, 1996, pp. 35–67.

[15] A. Fiat, *Batch RSA*, Advances in Cryptology – Crypto '89, Lecture Notes in Computer Science vol. 435, Springer Verlag, 1990, pp. 175–185.

[16] A. Fiat and A. Shamir, *How to Prove Yourself*, of Advances in Cryptology - Crypto '86, Lecture Notes in Computer Science 263, Springer Verlag, 1987, pp. 641–654.

[17] A. Fiat and A. Shamir, *Method, Apparatus, and Article for Identification and Signature*, United States Patent 4,748,668 (5/31/88)

[18] O. Goldreich, *Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme*, Advances in Cryptology – Crypto '86, Lecture Notes in Computer Science 263, Springer Verlag, 1987, pp. 104–110.

[19] O. Goldreich S. Goldwasser and S. Halevi, TR96-042, Electronic Colloquium on Computational Complexity, http://www.eccc.uni-trier.de/eccc/index.html.

[20] S. Goldwasser, S. Micali, and R. Rivest, *A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks*, SIAM J. Computing 17(2), pp. 281–301, 1988.

[21] R. Impagliazzo and M. Naor, *Efficient Cryptographic Schemes Provably as Secure as Subset-Sum*, J. of Cryptology, vol 9, 1996, pp. 199–216.

[22] R. Merkle, *A Digital Signature Based on a Conventional Encryption Function*, Advances in Cryptology – Crypto '87, Lecture Notes in Computer Science vol. 293, Springer Verlag, 1988, pp. 369–378.

[23] R. C. Merkle and M. Hellman, *Hiding information and Signature in Trapdoor Knapsack*, IEEE Transaction on Information Theory, Vol 24, 1978, pp. 525–530.

[24] S. Micali and A. Shamir, *An Improvement of the Fiat-Shamir Identification and Signature Scheme*, Proc. Advances in Cryptology – Crypto '88, Lecture Notes in Computer Science 403, Springer-Verlag, 1990, pp. 244–247.

[25] M. Naor and M. Yung, *Universal One Way Hash Functions and Their Cryptographic Applications*, Proc. 21st ACM Annual Symposium on the Theory of Computing, 1989, pp. 33–43.

[26] Lotus Notes *Internals* online book

[27] D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. Advances in Cryptology - Eurocrypt '96 Lecture Notes in Computer Science, Vol. 1070, Springer Verlag, 1996, pp. 387–398.

[28] M. O. Rabin *Digital Signatures and Public Key Functions as Intractable as Factoring*, Technical Memo TM-212, Lab. for Computer Science, MIT, Jan. 1979.

[29] R. Rivest, *The MD5 message digest algorithm*, RFC 1321, April 1991.

[30] R. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signature and Public Key Cryptosystems*, Comm. of ACM, 21 (1978), pp. 120–126.

[31] J. Rompel, *One-way Function are Necessary and Sufficient for Signatures*, Proc. 22nd ACM Annual Symposium on the Theory of Computing, 1990, pp. 387–394.

[32] C. P. Schnorr, *Efficient Signature Generation by Smart Cards*, J. of Cryptology **4**, pp. 161–174, 1991.

[33] C. P. Schnorr and H. H. Hörner, *Attacking the Chor-Rivest Cryptosystem by Improved Lattice Reduction* Proc. Advances in Cryptology – Eurocrypt'95, Lecture Notes in Computer Science vol. 925, 1995, pp. 1–12.

[34] A. Shamir, *On the Generation of Cryptographically Strong Pseudo-Random Number Sequences*, ACM Trans. Comput. Sys., 1 (1983), pp. 38–44.