A Forward-Secure Digital Signature Scheme

Mihir Bellare and Sara K. Miner

Dept. of Computer Science, & Engineering
University of California at San Diego, 9500 Gilman Drive
La Jolla, CA 92093, USA
{mihir,sminer}@cs.ucsd.edu
URL: http://www-cse.ucsd.edu/users/{mihir,sminer}

Abstract. We describe a digital signature scheme in which the public key is fixed but the secret signing key is updated at regular intervals so as to provide a *forward security* property: compromise of the current secret key does not enable an adversary to forge signatures pertaining to the past. This can be useful to mitigate the damage caused by key exposure without requiring distribution of keys. Our construction uses ideas from the Fiat-Shamir and Ong-Schnorr identification and signature schemes, and is proven to be forward secure based on the hardness of factoring, in the random oracle model. The construction is also quite efficient.

1 Introduction

This paper presents a digital signature scheme having a novel security property, namely "forward security." This is a means to mitigate the damage caused by key exposure.

1.1 The Key Exposure Problem

In practice the greatest threat against the security of a digital signature scheme is exposure of the secret (signing) key, due to compromise of the security of the underlying system or machine storing the key. The danger of successful cryptanalysis of the signature scheme itself is hardly as great as the danger of key exposure, as long as we stick to well-known schemes and use large security parameters.

The most widely considered solution to the problem of key exposure is distribution of the key across multiple servers via secret sharing [18,5]. There are numerous instantiations of this idea including threshold signatures [7] and proactive signatures [14]. Distribution however is quite costly. While a large corporation or a certification authority might be able to distribute their keys, the average user, with just one machine, does not have this option. Thus while we expect digital signatures to be very widely used, we do not expect most people to have the luxury of splitting their keys across several machines. Furthermore even when possible, distribution may not provide as much security as one might imagine. For example, distribution is susceptible to "common-mode failures:" a

Michael Wiener (Ed.): CRYPTO'99, LNCS 1666, pp. 431–448, 1999.

system "hole" that permits break-in might be present on all the machines since they are probably running a common operating system, and once found, all the machines can be compromised.

Other ways of protecting against key exposure include use of protected hardware or smartcards, but these can be costly or impractical.

1.2 Forward Secure Signatures

The goal of forward security is to protect some aspects of signature security against the risk of exposure of the secret signing key, but in a simple way, in particular without requiring distribution or protected storage devices, and without increasing key management costs.

How is it possible to preserve any security in the face of key exposure without distribution or protected devices? Obviously, we cannot hope for total security. Once a signing key is exposed, the attacker can forge signatures. The idea of "forward security" is however that a distinction can be made between the security of documents pertaining to (meaning dated in) the past (the time prior to key exposure) and those pertaining to the period after key exposure.

THE KEY EVOLUTION PARADIGM. A user begins, as usual, by registering a public key PK and keeping private the corresponding secret key, which we denote SK_0. The time during which the public key PK is desired to be valid is divided into periods, say T of them, numbered $1, \ldots, T$. While the public key stays fixed, the user "evolves" the secret key with time. Thus in each period, the user produces signatures using a different signing key: SK_1 in period 1, SK_2 in period 2, and so on. The secret key in period i is derived as a function of the one in the previous period; specifically, when period i begins, the user applies a public one-way function h to SK_{i-1} to get SK_i. At that point, the user also deletes SK_{i-1}. Now an attacker breaking in during period i will certainly get key SK_i, but not the previous keys $SK_0, \ldots, SK_{i-1}$, since they have been deleted. (Furthermore the attacker cannot obtain the old keys from SK_i because the latter was a one-way function of the previous key.) The key evolution paradigm is illustrated in Figure 1.

A signature always includes the value j of the time period during which it was produced, so that it is viewed as a pair $\langle j, \zeta \rangle$. The verification algorithm takes the (fixed) public key PK, a message and candidate signature, and verifies that the signature is valid in the sense that it was produced by the legitimate user *in the period indicated in the signature*. We stress that although the user's secret key evolves with time, the public key stays fixed throughout, so that the signature verification process is unchanged, as are the public key certification and management processes.

The number of periods and the length of each period are parameters of choice. For example, we might want to use the scheme under a certain public key for one year, with daily updates, in which case $T = 365$ and each period has length one day.

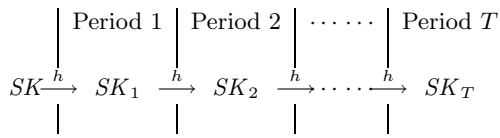

Fig. 1. Paradigm for a forward secure signature scheme: Secret key in a given period is a one-way function h of the secret key in the previous period.

Security benefits. The security benefit of this key evolution paradigm is that loss of the current secret key will not enable the adversary to forge signatures with a "date" prior to the one at which key exposure occurred. This "date" has a precise meaning: it is the value of the period during which the signature is (claimed to be) produced, which as indicated above is always included in the signature itself. This can be viewed as a means of protecting the authenticity of past transactions even in the event of key exposure. It is quite a useful property and can mitigate the damage of key exposure in several ways.

The following example, based on an idea due to Diffie, illustrates. Suppose Eve's mortgage payment is due on the first of each month. When it is delivered to the bank, the bank issues a signed and dated receipt. First suppose we are using a standard (not forward-secure) signature scheme. Then for example, if Eve paid \$1,000 on January 1st 1999, the text "Eve paid \$1,000 on January 1st, 1999" is signed under the public key of the bank. On February 1st, Eve is broke and doesn't pay, and of course the bank issues no receipt. But on February 2nd, Eve cracks the bank computer. Under this normal signing scheme, she gets the signing key, and can forge a note saying "Eve paid \$1,000 on February 1st, 1999". But not under a forward secure scheme. Under a forward secure scheme (assuming daily key updates) the signature that would have been produced on February 1st would have the form $\langle 32, \zeta \rangle$ where ζ is a tag for the text "Eve paid \$1,000," and we are assuming for simplicity that the scheme is initialized on January 1st, so that February 1st, being the 32nd day of the year, has corresponding date or period the number "32". Eve gets only the signing key for February 2nd, the 33rd period. She could forge any signature of the form $\langle j, \zeta \rangle$ for $j \geq 33$, for any text of her choice, but not one of the form $\langle 32, \zeta \rangle$. So she cannot claim to have paid up on February 1st.

Relation to time-stamping. Time-stamping signed documents via a trusted time stamping authority (cf. [13]) can also provide a similar kind of security, but this requires that one make use of such an authority, which is costly in various ways. Forward security may be viewed as providing a certain kind of time-stamp, namely one that is secure against forgery by an adversary who obtains the current secret key. (However it assumes the signer is honest since the signer could of course "forge" time-stamps if it wanted by simply not deleting previous keys.)

HISTORY. The term (perfect) "forward secrecy" was first used in [12] in the context of session key exchange protocols, and later in [8]. The basic idea, as described in [8], is that compromise of long-term keys does not compromise past session keys, meaning that past actions are protected in some way against loss of the current key, the same basic idea as here in a different context. The above paradigm and the idea of a digital scheme with forward security were suggested by Ross Andersen in an invited lecture at the ACM CCS conference [1]. He left as an open problem to find such a scheme. In this paper we provide the first solution. We also provide a formal adversarial model and notion of security with respect to which the security of schemes can be assessed. In particular we suggest the inclusion of the current date in the signature and the use of the claimed date as the determinant of whether something is past or present from the forward security point of view.

1.3 Construction of a Forward Secure Digital Signature Scheme

FIRST IDEAS. It is trivial to design a forward secure signature scheme if you allow any of scheme parameters to grow proportionally with the number T of time periods over which the scheme is supposed to be valid; specifically, if the size of the public key, secret key, or the signature itself is allowed to be proportional to T. We briefly describe these methods in Section 2. However, any such method is impractical.

The first reasonable solution, also described in Section 2, involves the use of binary tree based certification chains, and results in secret keys and signatures of size linear in $\lg(T)$. Ideally however we would like a solution in which the size of the public key, secret key, and signature does not depend on the number of time periods. But achieving forward security in this setting does not seem so easy. A natural starting idea is to try to find a suitable key evolution mechanism for standard RSA or El Gamal type schemes, but we were unable to find a secure one in either case.

OUR SCHEME. To get a forward-secure digital signature scheme of the desired kind, we turn to a different paradigm: construction of signature schemes based on identification schemes. We present a forward secure digital signature scheme based on ideas underlying the Fiat-Shamir [10] and Ong-Schnorr [16] identification and signature schemes. The feature of these schemes that is crucial to enable secure key evolution is that even the signer does not need to know the factorization of the modulus on which the scheme is based, but just the square roots of some public values. We evolve the secret key via squaring, which is a one-way function in this setting.

NOTION OF SECURITY. To provide assurance that our scheme has the forward security property, we first provide a formal definition of forward security for digital signatures, extending the notion of security for standard digital signatures from [11] to allow for key exposure attacks. Our model allows the adversary to mount a chosen-message attack, then expose the signing key at some current period j of its choice. It is successful if it can forge a signature of the form $\langle i, \zeta \rangle$

for some message M where $i < j$ pertains to a period prior to that of the key exposure. The scheme is secure if this task is computationally infeasible.

FORWARD SECURITY OF OUR SCHEME. We then show that our scheme meets this notion of forward security assuming it is hard to factor Blum-Williams integers. This proof is in the random oracle model [3], meaning it assumes that a certain hash function used in the scheme has random behavior.

Our security analysis proceeds in two steps. We first define a notion of a "forward secure identification (ID) scheme," design such a scheme, and prove it secure based on the hardness of factoring Blum-Williams integers. Our signature scheme is obtained from the ID scheme by the usual transformation of ID schemes into signature schemes (in which the challenge is specified as a hash of the message and the commitment). We then show that this transformation preserves forward security. (The transformation is known to preserve security in the standard sense [17,15], but here we are considering a new security feature.)

We stress one issue with regard to forward secure ID schemes: they are artificial constructs, in the sense that the security notion we put forth there, although mathematically viable, does not correspond to any real attack in a practical setting. (This is because identification is an on-line activity, unlike signature verification; one cannot return to the past and try to identify oneself.) But this does not matter in our setting, because for us, the ID scheme is simply a convenient abstraction that enables our construction of a forward secure signature scheme, and the latter is a real and useful object.

Our goal here has been to present the simplest possible scheme that has the forward security property, is reasonably efficient, and can be proven secure. Improvements and alternatives seem possible. In particular one could try to build a scheme only in the Ong-Schnorr style (rather than a combination of that with Fiat-Shamir as we do). This will reduce key sizes, but increase computation time, and the analysis (one could try to extend [19]) seems more involved.

A NOTE ON SYNCHRONIZATION. One might imagine that a scheme using time periods in the way we do will impose a requirement for clock synchronization, arising from disagreements near the time period boundaries over what is truly the current time. However, in our signature scheme, discrepancies over time periods do not cause problems. This is due to the fact that the time period in which the message was signed is stamped on the message, and the verifier looks at this stamp to determine which verification process to use. That is, it doesn't matter what the time period of verification is; the signature is verified using the information from the time period when the signature actually *occurred*.

2 Some Simple Solutions

We summarize several simple ways to design a forward secure signature scheme. The first three methods result in impractical schemes: some parameter (the public key size, the secret key size, or the signature size) grows linearly with the number T of time periods over which the public key is supposed to be valid. The

methods are nonetheless worth a brief look in order to better understand the problem and to lead into the fourth method. The latter, which we call the tree scheme, is a reasonable binary certification tree based solution in which key and signature sizes are logarithmic in T. Nonetheless it would be preferable if even this dependency is avoided, which is accomplished by our main scheme presented in Section 3.

In the following we make use of some fixed standard digital signature scheme whose signing and verifying algorithms are denoted SGN and VF respectively.

LONG PUBLIC AND SECRET KEYS. The signer generates T pairs $(p_1, s_1), \ldots, (p_T, s_T)$ of matching public and secret keys for the standard scheme. He sets the public key of the key evolving scheme to $(p_1, \ldots, p_T)$ and his starting (base) secret key for the key evolving scheme to $(s_0, s_1, \ldots, s_T)$ where s_0 is the empty string. Upon entering period j the signer deletes s_{j-1}, so that he is left with key $(s_j, \ldots, s_T)$. The signature of a message m in period j is $\langle j, \mathrm{SGN}_{s_j}(m)\rangle$. A signature $\langle j, \zeta\rangle$ on a message m is verified by checking that $\mathrm{VF}_{p_j}(m, \zeta) = 1$. This method clearly provides forward security, but the size of the keys (both public and secret) of the key evolving scheme depends on T, which is not desirable.

LONG SECRET KEY ONLY. Andersen [1] suggested the following variant which results in a short public key but still has a secret key of size proportional to T. Generate T key-pairs as above, and an additional pair (p, s). Let $\sigma_j = \mathrm{SGN}_s(j \| p_j)$ be a signature (with respect to p) of the value j together with the j-th public key, for $j = 1, \ldots, T$. This done, delete s. The public key of the key evolving scheme is p, and the signer's starting (base) secret key for the key evolving scheme is $(s_0, \sigma_0, s_1, \sigma_1, \ldots, s_T, \sigma_T)$ where s_0, σ_0 are set to the empty string. Upon entering period j the signer deletes s_{j-1}, σ_{j-1}, so that he is left with secret key $(s_j, \sigma_j, \ldots, s_T, \sigma_T)$. The signature of a message m in period j is $\langle j, (\mathrm{SGN}_{s_j}(m), p_j, \sigma_j)\rangle$. A signature $\langle j, (\alpha, q, \sigma)\rangle$ on a message m is verified by checking that $\mathrm{VF}_q(m, \alpha) = 1$ and $\mathrm{VF}_p(j \| q, \sigma) = 1$. This method, like the above, clearly provides forward security. (Notice that it is crucial for the forward security that s be deleted as soon as the signatures of the subkeys have been generated.) Furthermore the public key has size independent of T. But the size of the secret key still depends on T.

LONG SIGNATURES. The size of both the public and the secret key can be kept small by using certification chains, but this results in large signatures. The signer begins by generating a key pair (p_0, s_0) of the standard scheme. He sets the public key of the key evolving scheme to p_0 and the starting (base) secret key of the key evolving scheme to s_0. At each period a new key is generated and certified with respect to the previous key, which is then deleted. The certificate chain is included in the signature. To illustrate, at the start of period 1 the signer creates a new key pair (p_1, s_1), sets $\sigma_1 = \mathrm{SGN}_{s_0}(1 \| p_1)$, and deletes s_0. The signature of a message m in period 1 is $\langle 1, (\mathrm{SGN}_{s_1}(m), p_1, \sigma_1)\rangle$. A signature $\langle 1, (\alpha, q_1, \tau_1)\rangle$ on a message m is verified by checking that $\mathrm{VF}_{q_1}(m, \alpha) = 1$ and $\mathrm{VF}_{p_0}(1 \| q_1, \tau_1) = 1$. This continues iteratively, so that at the start of period $j \geq 2$ the signer, in possession of $p_1, \sigma_1, \ldots, p_{j-1}, \sigma_{j-1}$ and the secret key s_{j-1} of the previous period,

creates a new key pair (p_j, s_j), sets $\sigma_j = \mathrm{SGN}_{s_{j-1}}(j \| p_j)$, and deletes s_{j-1}. The signature of a message m in period j is $\langle j, (\mathrm{SGN}_{s_j}(m), p_1, \sigma_1, \ldots, p_j, \sigma_j)\rangle$. A signature $\langle j, (\alpha, q_1, \tau_1, \ldots, q_j, \tau_j)\rangle$ on a message m is verified by checking that $\mathrm{VF}_{q_j}(m, \alpha) = 1$ and $\mathrm{VF}_{q_{i-1}}(i \| q_i, \tau_i) = 1$ for $i = 2, \ldots, j$ and $\mathrm{VF}_{p_0}(1 \| q_1, \tau_1) = 1$. Again, forward security is clearly provided. Furthermore both the public and the secret key are of size independent of T. But the size of a signature grows linearly with T. Also note that although the size of the signer's secret key has shrunk, the signer does have to store the list of public keys and their tags, which means it must use storage linear in T, which is not desirable.

Binary certification tree scheme. Andersen's scheme above can be viewed as building a certification tree of depth 1 and arity T, while the "long signature" solution just presented builds a standard certification chain, which is a tree of depth T and arity 1. Signature size is linear in the depth, and key size is linear in the arity. It is natural to form a "hybrid" scheme so as to get the best possible tradeoff between these sizes. Namely we use a binary tree with T leaves. Each key certifies its two children keys. The leaf keys are the actual message signing keys, one for each period, and the root public key is the public key of the key evolving scheme. The tree grows down from the root in a specific manner, with nodes being created dynamically. A node certifies its two children, and as soon as the two children nodes are created and certified, the secret key corresponding to the parent node is deleted. Appropriately done this results in a forward secure scheme with secret key size and signature size linear in $\lg(T)$, and fixed public key size. Furthermore the total amount of information (whether secret or not) stored at any time by the signer, and the computation time for key updates, are linear in $\lg(T)$.

One has to be a little careful to build the tree in the right way and delete keys at the right times, so let us give a few more details. For simplicity assume $T = 2^\tau$ is a power of two. Imagine a binary tree with T leaves. The root is at level 0 and the leaves are at level τ. The root is labeled with the empty string ε, and if a node has label the binary string w then its left child is labeled $w0$ and its right child is labeled $w1$. Associate to a node with label w a key pair $(p[w], s[w])$. (The entire tree never exists at any time, but is useful to imagine.) The public key of the key evolving scheme is $p[\varepsilon]$. Let $\langle i, n\rangle$ denote the binary representation of the integer $i - 1$ as a string of exactly n bits.

In period $j \geq 1$ the signer signs data under $s[\langle j, \tau\rangle]$, and attaches a certification chain based on the path from leaf $\langle j, \tau\rangle$ to the root of the tree. During period j, a certain subtree $\mathcal{T}$ of the full tree exists. It consists of all nodes on the path from leaf $\langle j, \tau\rangle$ to the root, plus the right sibling of any of these nodes that is a left child of its parent. All childless nodes w in the subtree that are right children of their parents have their secret key $s[w]$ still attached; for all other nodes, it has been deleted. Notice that if a node w has $s[w]$ still "alive," its descendent leaves have names $\langle i, \tau\rangle$ with $i > j$, so correspond to future periods. When the signer enters period $j + 1$ (here $j < T$) it must update its subtree. This involves (possibly) creation of new nodes, and deletion of old ones in such a way that the property of the subtree described above is preserved. The signer

moves up from $\langle j, \tau \rangle$ (deleting this leaf node and its keys) and stops at the first node whose left (rather than right) child is on the path from $\langle j, \tau \rangle$ to the root. Call this node w. We know that the secret key of its right child $w1$ is alive. If $w1$ is a leaf, the update is complete. Else, create its children by picking new key pairs for each child, and delete the secret key at w. Then move left, and continue this process until a leaf is reached.

One can show that the scheme has the desired properties.

3 Our Forward-Secure Signature Scheme

KEYS AND KEY GENERATION. The signer's public key contains a modulus N and l points $U_1, \ldots, U_l$ in Z_N^*. The corresponding *base secret key* SK_0 contains points $S_1, \ldots, S_l$ in Z_N^*, where S_j is a 2^{T+1}-th root of U_j for $j = 1, \ldots, T$. The signer generates the keys by running the following key generation process, which takes as input the security parameter k determining the size of N, the number l of points in the keys, and the number T of time periods over which the scheme is to operate.

Algorithm $KG(k, l, T)$
Pick random, distinct $k/2$ bit primes p, q, each congruent to 3 mod 4
$N \leftarrow pq$
For $i = 1, \ldots, l$ **do** $S_i \stackrel{R}{\leftarrow} Z_N^*$; $U_i \leftarrow S_i^{2^{(T+1)}} \bmod N$ **EndFor**
$SK_0 \leftarrow (N, T, 0, S_{1,0}, \ldots, S_{l,0})$; $PK \leftarrow (N, T, U_1, \ldots, U_l)$
Return (PK, SK_0)

As the code indicates, the keys contain some sundry information in addition to that mentioned above. Specifically the number T of time periods is thrown into the public key. This enables the verifier to know its value, which might vary with different signers. It is also thrown into the secret key for convenience, as is the modulus N. The third component of the base secret key, namely "0", is there simply to indicate that this is the base key, in light of the fact that the key will be evolving later. The modulus is a Blum-Williams integer, meaning the product of two distinct primes each congruent to 3 mod 4. We refer to U_i as the i-th *component* of the public key.

The public key PK is treated like that of any ordinary signature scheme as far as registration, certification and key generation are concerenced. The base secret key SK_0 is stored privately. The factors p, q of N are deleted once the key generation process is complete, so that they are not available to an attacker that might later break into the system on which the secret key is stored.

KEY EVOLUTION. During time period j, the signer signs using key SK_j. This key is generated at the start of period j, by applying a key update algorithm to the key SK_{j-1}. (The latter is the base secret key when $j = 1$.) The update algorithm is presented below. It squares the l points of the secret key at the previous stage to get the secret key at the next stage.

Algorithm $Upd(SK_{j-1}, j)$ where $SK_{j-1} = (N, T, j-1, S_{1,j-1}, \ldots, S_{l,j-1})$
  For $i = 1, \ldots, l$ **do** $S_{i,j} \leftarrow S_{i,j-1}^2 \bmod N$ **EndFor**
  $SK_j \leftarrow (N, T, j, S_{1,j}, \ldots, S_{l,j})$
  Return SK_j

Once this update is performed, the signer deletes the key SK_{j-1}. Since squaring modulo N is a one-way function when the factorization of N is unknown, it is computationally infeasible to recover SK_{j-1} from SK_j. Thus key exposure during period j will yield to an attacker the current key (and also future keys) but not past keys. We refer to $S_{i,j}$ as the *i-th component of the period j secret key*. Notice that the components of the secret key for period j are related to those of the base key as follows:

$$(S_{1,j}, \ldots, S_{l,j}) \;=\; (S_{1,0}^{2^j}, \ldots, S_{l,0}^{2^j})\,. \tag{1}$$

We will use this later.

The length of a time period (during which a specific key is in use) is assumed to be globally known. The choice depends on the application and desired level of protection against key exposure; it could vary from seconds to days.

SIGNING. Signature generation during period j is done via the following algorithm, which takes as input the secret key SK_j of the current period, the message M to be signed, and the value j of the period itself, to return a signature $\langle j, (Y, Z)\rangle$, where Y, Z are values in Z_N^*. The signer first generates the "committment" Y, and then hashes Y and the message M via a public hash function $H\colon \{0,1\}^* \to \{0,1\}^l$ to get an l-bit "challenge" $c = c_1 \ldots c_l$ which is viewed as selecting a subset of the components of the public key. The signer then computes a 2^{T+1-j}-th root of the product of the public key components in the selected subset, and multiplies this by a 2^{T+1-j}-th root of Y to get the value Z. This is detailed in the signing algorithm below.

Algorithm $Sgn_{SK_j}^H(M, j)$ where $SK_j = (N, T, j, S_{1,j}, ...S_{l,j})$
  $R \stackrel{R}{\leftarrow} Z_N^*$; $Y \leftarrow R^{2^{(T+1-j)}} \bmod N$; $c_1 \ldots c_l \leftarrow H(j, Y, M)$
  $Z \leftarrow R \cdot \prod_{i=1}^{l} S_{i,j}^{c_i} \bmod N$
  Output $\langle j, (Y, Z)\rangle$

Thus, in the first time period, the signer is computing 2^T-th roots; in the second time period, 2^{T-1}-th roots; and so on, until the last time period, where it is simply computing square roots. Notice that in the last time period, we simply have the Fiat-Shamir signature scheme. (The components of the secret key SK_T are square roots of the corresponding components of the public key.) The hash function $H\colon \{0,1\}^* \to \{0,1\}^l$ will be assumed in the security analysis to be a random oracle.

Notice that the index j of the period during which the signature was generated is part of the signature itself. This provides some sort of "time-stamp", or claimed time-stamp, and is a crucial element of the scheme and model, since security will pertain to the ability to generate signatures having such "time-stamps" with value earlier than the current date.

VERIFYING. Verification of a candidate signature $\langle j, (Y, Z)\rangle$ for a given message M with respect to a given public key PK is performed via the following process:

Algorithm $Vf^H_{PK}(M, \langle j, (Y, Z)\rangle)$ where $PK = (N, T, U_1, ..., U_l)$
$c_1 \ldots c_l \leftarrow H(j, Y, M)$
If $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$ **Then** return 1 **Else** return 0

Note the verification process depends on the claimed "time-stamp" or period indicator j in the signature, meaning that the period j too is authenticated.

The signature scheme is fully specified by the above four algorithms. We let $\mathsf{FSIG}[k, l, T]$ denote our scheme when the parameters are fixed as indicated.

COST. The scheme FSIG is quite efficient. Signing in period j takes $T+1-j+l/2$ multiplications modulo N on the average. (So the worst case cost does not exceed $T + 1 + l$ multiplications, and in fact the scheme gets faster as time progresses.) For typical values of the parameters, this can be less than the cost of a single exponentiation, making signing cheaper than in RSA based schemes. Verification has the same cost as signing. (We ignore the cost of hashing, which is lower than that of the modular arithmatic.)

Like the Fiat-Shamir scheme, however, the key sizes are relatively large, being proportional to l. The size of the public key can be reduced by having its components $U_1, \ldots, U_l$ specified implicitly rather than explicitly, as values of a random-oracle hash funcion on some fixed points. That is, the signer can choose some random constant U, say 128 bits long, and then specify small values $a_1, \ldots, a_l$ such that $H^*(U, a_i)$ is a square modulo N, where H^* is a hash function with range Z_N^*. The public key is now $(N, T, U, a_1, \ldots, a_l)$. Since N is a Blum-Williams integer, the signer can then compute a 2^{T+1}-th root of $u_i = H(U, a_i)$, and thereby have a secret key relating to the public key in the same way as before. The average size of the list $a_1, \ldots, a_l$ is very small, about $O(l \lg(l))$ bits. Unfortunately it is not possible to similarly shrink the size of the secret key in this scheme, but moving to post-Fiat-Shamir identification-derived signature schemes, one can get shorter keys.

VALIDITY OF GENUINE SIGNATURES. Before we discuss security, we should check that signatures generated by the signing process will always be accepted by the verification process.

Proposition 1. *Let* $PK = (N, T, U_1, \ldots, U_l)$ *and* $SK_0 = (N, T, 0, S_{1,0}, \ldots, S_{l,0})$ *be a key pair generated by the above key generation algorithm. Let* $\langle j, (Y, Z)\rangle$ *be an output of* $Sgn^H_{PK}(M, j)$. *Then* $Vf_{PK}(M, \langle j, (Y, Z)\rangle) = 1$.

Proof: Let $c_1 \ldots c_l \leftarrow H(j, Y, M)$. We check that $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$ using Equation (1) and the fact that the signature was correctly generated:

$$\begin{aligned} Z^{2^{(T+1-j)}} &= \left(R \cdot \prod_{i=1}^{l} S_{i,j}^{c_i}\right)^{2^{(T+1-j)}} \bmod N \\ &= R^{2^{(T+1-j)}} \cdot \prod_{i=1}^{l} S_{i,0}^{c_i \cdot 2^{j+(T+1-j)}} \bmod N \end{aligned}$$

$$= Y \cdot \prod_{i=1}^{l} S_{i,0}^{c_i \cdot 2^{(T+1)}} \bmod N$$
$$= Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N \ ,$$

as desired. ▮

SECURITY MODEL. We wish to assess the forward security of our scheme. To do this effectively we must first pin down an appropriate model; what can the adversary do, and when is it declared successful?

Recall the goal is that even under exposure of the current secret key it should be computationally infeasible for an adversary to forge a signature "with respect to" a previous secret key. The possibility of key exposure is modeled by allowing the adversary a "break-in" move, during which it can obtain the secret key SK_j of the current period j. The adversary is then considered successful if it can create a valid forgery where the signature has the form $\langle i, (Y, M)\rangle$ for some $i < j$, meaning is dated for a previous time period. The model is further augmented to allow the adversary a chosen-message attack prior to its break-in. In that phase, the adversary gets to obtain genuine signatures of messages of its choice, under the keys $SK_1, SK_2, \ldots$ in order, modeling the creation of genuine signatures under the key evolution process. The adversary stops the chosen-message attack phase at a point of its choice and then gets to break-in. Throughout the adversary is allowed oracle access to the hash function H since the latter is modeled as a random oracle.

Thus the adversary F is viewed as functioning in three stages: the chosen message attack phase (cma); the break-in phase (breakin); and the forgery phase (forge). Its success probability in breaking $\mathsf{FSIG}[k, l, T]$ is evaluated by the following experiment:

Experiment F-Forge($\mathsf{FSIG}[k, l, T], F$)
Select H: $\{0,1\}^* \to \{0,1\}^l$ at random
$(PK, SK_0) \xleftarrow{R} KG(k, l, T)$
$i \leftarrow 0$
`Repeat`
$j \leftarrow j + 1$; $SK_j \leftarrow Upd(SK_{j-1}, j)$; $d \leftarrow F^{H, Sgn^H_{SK_j}(\cdot, j)}(\mathsf{cma}, PK)$
`Until` $(d = \mathsf{breakin})$ or $(j = T)$
$(M, \langle b, \zeta\rangle) \leftarrow F^H(\mathsf{forge}, SK_j)$
`If` $Vf^H_{PK}(M, \langle b, \zeta\rangle) = 1$ and $1 \le b < j$ and M was not queried of $Sgn_{SK_b}(\cdot, b)$
`then return` 1 `else return` 0

It is understood above that in running F we first pick and fix coins for it, and also that we preserve its state across its various invocations. The chosen-message attack reflects the way the signature scheme is used. The adversary first gets access to an oracle for generating signatures under SK_1. It queries this as often as it wants, and indicates it is done by outputting some value d. As long as d is not the special value breakin, we move into the next stage, providing the adversary an oracle for signing under the next key. Note that the process is

strictly ordered; once an adversary gives up the oracle for signing under SK_j, by moving into the next phase or breaking-in, it cannot obtain access to that oracle again. At some point the adversary decides to use its break-in privilege, and is returned the key SK_j of the stage in which it did this. (If it does not break-in by the last period, we give it the key of that period by default.) It will now try to forge signatures under SK_b for some $b < j$ and is declared successful if the signature is valid and the message is new.

Following the concrete security paradigm used in [4], we associate to the scheme an *insecurity function* whose value is the maximum probability of being able to break the scheme, the maximum being over all adversary strategies restricted to resource bounds specified as arguments to the insecurity function. To make this precise we begin by letting $\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[k,l,T],F)$ denote the probability that the above experiment returns 1. (This is the probability that the adversary F is successful in breaking FSIG in the forward security sense.) Then the insecurity function is defined as

$$\mathbf{InSec}^{\mathrm{fwsig}}(\mathsf{FSIG}[k,l,T];t,q_{\mathrm{sig}},q_{\mathrm{hash}}) \;=\; \max_{F}\,\{\,\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[k,l,T],F)\,\}\;.$$

The maximum here is over all F for which the following are true; the execution time of the above experiment, plus the size of the code of F, is at most t; F makes a total of at most q_{sig} queries to the signing oracles across all the stages; the total number of queries made to H in the execution of the experiment is at most q_{hash}.

Note the execution time is not just that of F but rather that of the entire experiment F-Forge($\mathsf{FSIG}[k,l,T],F$), so includes in particular the time to compute answers to oracle queries. The time to pick the hash function H is also included, measured dynamically by counting the time to reply to each hash oracle query by picking the response randomly at the time the query is made. Similarly, q_{hash} is the number of H queries in the experiment, not just the number made explicitly by F, so that it includes the H queries made by the signing and verifying algorithms. In particular, q_{hash} is always at least one due to the hash query made in the verification of the forgery.

The smaller the value taken by the insecurity function, the more secure the scheme. Our goal will be to upper bound the values taken by this insecurity function.

FACTORING. We will prove the security of our scheme by upper bounding its insecurity as a function of the probability of being able to factor the underlying modulus in time related to the insecurity parameters. To capture this, let $Fct(\cdot)$ be any algorithm that on input a number N, product of two primes, attempts to return its prime factors, and consider the experiment:

Experiment Factor(k, Fct)
Pick random, distinct $k/2$ bit primes p, q, each congruent to 3 mod 4
$N \leftarrow pq$
$p', q' \leftarrow Fct(N)$
If $p'q' = N$ and $p' \neq 1$ and $q' \neq 1$ `then return` 1 `else return` 0

Let $\mathbf{Succ}^{\mathrm{fac}}(Fct, k)$ denote the probability that the above experiment returns 1. Let $\mathbf{InSec}^{\mathrm{fac}}(k, t)$ denote the maximum value of $\mathbf{Succ}^{\mathrm{fac}}(k, Fct)$ over all algorithms Fct for which the running time of the above experiment plus the size of the description of Fct is at most t. This represents the maximum probability of being able to factor a k-bit Blum-Williams integer in time t. We assume factoring is hard, meaning $\mathbf{InSec}^{\mathrm{fac}}(k, t)$ is very low as long as t is below the running time of the best known factoring algorithm, namely about $2^{1.9k^{1/3}\lg(k)^{2/3}}$.

Security of our signature scheme. We are able to prove that as long as the problem of factoring Blum-Williams integers is computationally intractable, it is computationally infeasible to break the forward security of our signature scheme. The following theorem provides a precise, quantitative statement, upper bounding the forward insecurity of our scheme as a function of the insecurity of factoring.

Theorem 1. *Let* $\mathsf{FSIG}[k, l, T]$ *represent our key evolving signature scheme with parameters modulus size* k*, hash function output length* l*, and number of time periods* T*. Then for any* t*, any* q_{sig}*, and any* $q_{\mathrm{hash}} \geq 1$

$$\begin{aligned}&\mathbf{InSec}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, l, T]; t, q_{\mathrm{sig}}, q_{\mathrm{hash}})\\ &\leq q_{\mathrm{hash}} \cdot T \cdot \left[2^{-l} + \sqrt{2lT \cdot \mathbf{InSec}^{\mathrm{fac}}(k, t')}\right] + \frac{q_{\mathrm{sig}} \cdot q_{\mathrm{hash}}}{2^k}\ ,\end{aligned}$$

where $t' = 2t + O(k^3 + k^2 l \lg(T))$.

The security parameter k must be chosen large enough that the $\mathbf{InSec}^{\mathrm{fac}}(k, t')$ (the probability of being able to factor the modulus in time t') is very low. The theorem then tells us that the probability of being able to compromise the forward security of the signature scheme is also low.

The theorem above proves the security of the scheme in a qualitative sense: certainly it implies that polynomial time adversaries have negligible advantage, which is the usual complexity based goal. However it also provides the concrete, or exact security, via the concrete indicated bound. In this case however, we know no attacks achieving a success matching our bound, and suspect our bound is not tight. Perhaps the concrete security can be improved, particularly with regard to the $q_{\mathrm{hash}} \cdot T$ multiplicative factor.

The proof of this theorem is in two steps. Section 4 makes explicit an identification scheme that underlies the above signature scheme, and Lemma 1 shows that this identification scheme is forward secure as long as factoring is hard. Section 5 relates the forward security of our signature scheme to that of the identification scheme via Lemma 2. Putting these two lemmas together easily yields Theorem 1. We now turn to the lemmas.

4 Our Forward-Secure Identification Scheme

Forward-secure identification. We consider the standard framework for a public-key based identification protocol. The prover is in possession of secret key

SK_0, while both the prover and the verifier are in possession of the corresponding public key PK. The prover wants to identify herself interactively to the verifier. A standard three-pass protocol will be used, consisting of a "commitment" Y from the prover, followed by a challenge c from the verifier, and finally an "answer" Z from the prover. The standard security concern is that an adversary (not in possession of the secret key) be able to identify itself to the prover under the public key PK of the legitimate prover.

We extend the setting to allow evolution of the secret key, enabling us to consider forward security. Thus the time over which the public key is valid is divided into T periods, and in period j the prover identifies itself using a secret key SK_j. The public key remains fixed, but the protocol and verification procedure depend on j, which takes the value of the current period, a value that all parties are agreed upon. Forward security means that an adversary in possession of SK_j should still find it computationally infeasible to identify itself to the verifier in a time period i previous to j.

This security measure having been stated should, however, at once create some puzzlement. Identification is an "on-line" activity. More specifically, verification is an on-line, one-time action. Once period i is over, there is no practical meaning to the idea of identifying oneself in period i; one cannot go back in time. So forward security is not a real security concern or attribute in identification. So why consider it? For us, forward-secure identification is only a convenient mathematical game or abstraction based on which we can analyze the forward security of our signature scheme. (And forward security of the signature scheme is certainly a real concern: the difference with identification is that for signatures verification can take place by anyone at any time after the signature is created.)

In other words, we can certainly define and consider a mathematically (if not practically) meaningful notion of forward security of an identification scheme, and analyze a scheme with regard to meeting this property. That is what we do here, for the purpose of proving Theorem 1.

THE SCHEME. Keys (both public and secret) are identical to those of our signature scheme, and are generated by the same procedure KG that we described in Section 3, so that at the start of the game the prover is in possession of the base secret key SK_0. The public key PK is assumed to be in possession of the verifier. At the start of period j ($1 \leq j \leq T$) the prover begins by updating the previous secret key SK_{j-1} to the new secret key SK_j that she will use in period j. This process is done by the same update algorithm as for the signature scheme, specified in Section 3. The update having been performed, the key SK_{j-1} is deleted. During period j, the prover and verifier may engage in any number of identification protocols with each other, as need dictates. In these protocols, it is assumed both parties are aware of the value j indicating the current period. The prover uses key SK_j to identify herself. We depict in Figure 2 the identification protocol in period j, showing the steps taken by both the prover and the verifier.

The identification scheme is fully specified by key generation algorithm, key update algorithm, and the proving and verifying algorithms that underly

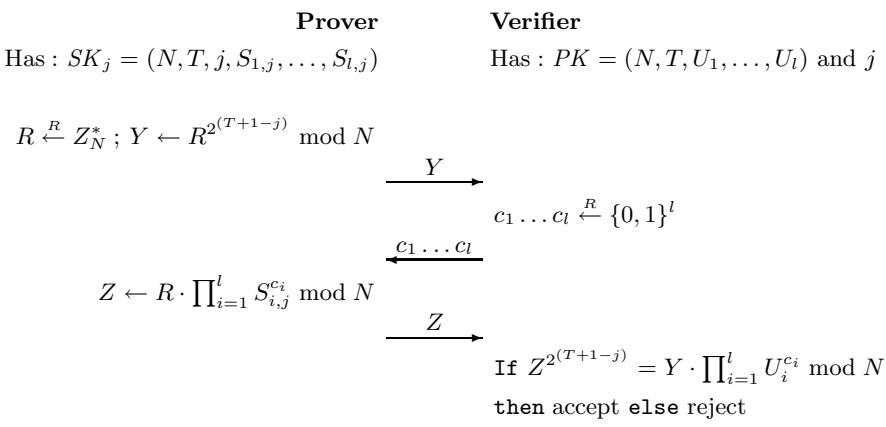

Fig. 2. Forward secure ID scheme: The protocol executed in period j. The prover has the secret key SK_j for period j and the verifier has the base public key PK and the value of j.

Figure 2. We let $\mathsf{FID}[k, l, T]$ denote our identification scheme when the parameters are fixed as indicated.

It should of course be the case that when the prover is honest, the verifier accepts. The proof of this is just like the proof of Proposition 1, the analogous issue for the signature scheme.

As should be clear from the identification protocol of Figure 2, the signing procedure under key SK_j that we defined in Section 3 is closely related to the identification protocol in period j: the former is derived from the latter by the standard process of specifying the challenge c via the value of a hash function on the message M and the commitment Y. This connection enables us to break the security analysis of the signature scheme into two parts; a forward-security analysis of the identification scheme, and an analysis of the preservation of forward-security of the "challenge hash" paradigm. This section is devoted to the first issue. We begin with the model.

Security model. The adversary in an identification scheme is called an impersonator and is denoted I. It knows the public key PK of the legitimate prover, and the current period j. We view I as functioning in two stages: the break-in phase ($\mathsf{breakin}$) and the impersonation phase (imp). Its success probability in breaking FID is determined by the following experiment:

Experiment F-Impersonate($\mathsf{FID}[k, l, T], I$)
$(PK, SK_0) \stackrel{R}{\leftarrow} KG(k, l, T)$
$b \leftarrow I(\mathsf{breakin}, PK)$
`For` $j = 1, \ldots, b$ `do` $SK_j \leftarrow Upd(SK_{j-1}, j)$; $j \leftarrow j + 1$ `EndFor`
$Y, j \leftarrow I(\mathsf{imp}, SK_b)$; $c_1 \ldots c_l \stackrel{R}{\leftarrow} \{0,1\}^l$; $Z \leftarrow I(c_1 \ldots c_l)$
`If` $Z^{2^{(T+1-j)}} = Y \cdot \prod_{i=1}^{l} U_i^{c_i} \bmod N$ and $j < b$
`then return` 1 `else return` 0

In the break-in phase the adversary returns a time period $b \in \{1, \ldots, T\}$ as a break-in time, as a function of the public key. It will be returned the corresponding secret key SK_b, and now will try to successfully identify itself relative to some period $j < b$ of its choice. Here it will try to impersonate the prover, so first chooses some commitment value Y. Upon being given the random challenge $c_1 \ldots c_l$ from the verifier, it returns an answer Z. It is considered successful if the verifier accepts the answer. Above, U_i is the i-th component of the public key PK. We let $\mathbf{Succ}^{\text{fwid}}(\mathsf{FID}[k, l, T], I)$ denote the probability that the above experiment returns 1 and then let

$$\mathbf{InSec}^{\text{fwid}}(\mathsf{FID}[k, l, T]; t) \;=\; \max_{I} \{\, \mathbf{Succ}^{\text{fwid}}(\mathsf{FID}[k, l, T], I) \,\} \;,$$

the maximum being over all I that for which the execution time of the above experiment, plus the size of the code of F, is at most t. As usual, the smaller the value taken by the insecurity function, the more secure the scheme. Our goal will be to upper bound the values taken by this insecurity function.

Notice that our model does not allow the adversary to first play the role of a cheating verifier, as does the standard model of identification. The reason is that we are only interested in applying this to the signature scheme, and there the challenge, being the output of the hash function, will be random, corresponding to an honest verifier. So it suffices to consider an honest verifier here.

Security of our identification scheme. Next, we prove the the security of our identification scheme by showing that breaking the forward security of our identification scheme is hard as long as the problem of factoring a Blum-Williams integer into its two prime factors is hard. The following lemma states a result which implies this:

Lemma 1. *Let* $\mathsf{FID}[k, l, T]$ *represent our key evolving identification scheme with modulus size* k*, challenge length* l*, and number of time periods* T*. Then for any* t

$$\mathbf{InSec}^{\text{fwid}}(\mathsf{FID}[k, l, T]; t) \leq 2^{-l} + \sqrt{2lT \cdot \mathbf{InSec}^{\text{fac}}(k, t')} \;,$$

where $t' = 2t + O(k^3 + k^2 l \lg(T))$.

The first term in the bound represents the probability that the impersonator guesses the verifier's challenge, in which case it can of course succeed. The second term rules out any other attacks that are very different from simply trying to factor the modulus. The proof of Lemma 1 is omitted due to lack of space, and can be found in the full version of this paper [2] which is available on the web.

5 From Identification to Signatures

As we have noted, our signature scheme is derived from our identification scheme in the standard way, namely by having the signer specify the challenge c as a hash of the message M and commitment Y. In the standard setting we know that

this paradigm works, in the sense that if the identification scheme is secure then so is the signature scheme [17,15]. However we are not in the standard setting; we are considering an additional security property, namely forward security. The previous results do not address this. It turns out however that the hashing paradigm continues to work in the presence of forward security, in the following sense: if the identification scheme is forward secure, so is the derived signature scheme. This is a consequence of the following lemma:

Lemma 2. *Let* $\mathsf{FID}[k, l, T]$ *and* $\mathsf{FSIG}[k, l, T]$ *represent, respectively our key evolving identification scheme and our key evolving signature scheme, with parameters* k, l, T*. Then for any* t*, any* $q_{\rm sig}$*, and any* $q_{\rm hash} \geq 1$

$$\mathbf{InSec}^{\rm fwsig}(\mathsf{FSIG}[k, l, T]; t, q_{\rm sig}, q_{\rm hash}) \leq q_{\rm hash} \cdot T \cdot \mathbf{InSec}^{\rm fwid}(\mathsf{FID}, t') + \frac{q_{\rm sig} \cdot q_{\rm hash}}{2^k},$$

where $t' = t + O(k^2 l \lg(T))$.

The lemma says that if the forward insecurity of the identification scheme is small, so is that of the signature scheme. Combining Lemmas 2 and 1 proves Theorem 1, the main theorem saying our signature scheme is forward secure. The proof of Lemma 2 is omitted due to lack of space, and can be found in the full version of this paper [2] which is available on the web.

Acknowledgments

We thank the Crypto 98 program committee and Michel Abdalla for their comments, and Victor Shoup for helpful discussions. Mihir Bellare was supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering. Sara Miner was supported in part by a National Physical Sciences Consortium Fellowship and the above-mentioned grants of Bellare.

References

1. R. Anderson, Invited lecture, *Fourth Annual Conference on Computer and Communications Security*, ACM, 1997.
2. M. Bellare and S. Miner, "A forward-secure digital signature scheme," Full version of this paper, available via `http://www-cse.ucsd.edu/users/mihir`.
3. M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.
4. M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rabin," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.
5. G. R. Blakley, "Safeguarding cryptographic keys." *Proceedings of AFIPS 1979 National Computer Conference*, AFIPS, 1979.
6. L. Blum, M. Blum and M. Shub, "A simple unpredictable pseudo-random number generator," *SIAM Journal on Computing* Vol. 15, No. 2, 364-383, May 1986.

7. Y. Desmedt and Y. Frankel, "Threshold cryptosystems." *Advances in Cryptology – Crypto* 89 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 435, G. Brassard ed., Springer-Verlag, 1989.
8. W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, 2, 107–125 (1992).
9. U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity," *J. of Cryptology*, 1(1988), 77-94.
10. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Advances in Cryptology – Crypto* 86 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986.
11. S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing,* Vol. 17, No. 2, pp. 281–308, April 1988.
12. C. Günther, "An identity-based key-exchange protocol," *Advances in Cryptology – Eurocrypt* 89 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 434, J-J. Quisquater, J. Vandewille ed., Springer-Verlag, 1989.
13. S. Haber and W. Stornetta, "How to Time-Stamp a Digital Document," *Advances in Cryptology – Crypto* 90 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 537, A. J. Menezes and S. Vanstone ed., Springer-Verlag, 1990.
14. A Herzberg, M. Jakobsson, S .Jarecki, H Krawczyk and M. Yung, "Proactive public key and signature schemes," *Proceedings of the Fourth Annual Conference on Computer and Communications Security*, ACM, 1997.
15. K. Ohta and T. Okamoto. "On concrete security treatment of signatures derived from identification," *Advances in Cryptology – Crypto* 98 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.
16. H. Ong and C. Schnorr, "Fast signature generation with a Fiat-Shamir like scheme," *Advances in Cryptology – Eurocrypt* 90 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 473, I. Damgård ed., Springer-Verlag, 1990.
17. D. Pointcheval and J. Stern, "Security proofs for signature schemes," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.
18. A. Shamir, "How to share a secret," *Communications of the ACM*, 22(1979), 612-613.
19. V. Shoup, "On the security of a practical identification scheme," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lec. Notes in Comp. Sci. Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.
20. H. Williams, "A Modification of the RSA Public-key Encryption Procedure," *IEEE Transactions on Information Theory*, Vol. IT-26, No. 6, 1980, pp. 726–729.