Simple Forward-Secure Signatures From Any Signature Scheme*

Hugo Krawczyk†

Abstract

In Crypto'99, Bellare and Miner introduced *forward-secure signatures* as digital signature schemes with the attractive property that exposure of the signing key at certain time period does not allow for the forgery of signatures from previous time periods. That paper presented the first full design of an efficient forward-secure signatures scheme, but left open the question of building efficient and practical schemes based on standard signatures such as RSA or DSS. In particular, they called for the development of schemes where the main size-parameters (namely, the size of the private key, public key, and signature) do not grow with the total number of periods for which the public key is to be in use.

We present an efficient and extremely simple construction of forward-secure signatures based on *any* regular signature scheme (e.g., RSA and DSS); the resultant signatures enjoy size-parameters that are independent of the number of periods (except for the inclusion of an index to the period in which a signature is issued). The only parameter that grows (linearly) with the number of periods is the total size of local non-secret memory of the signer. The forward-security of our schemes is directly implied by the unforgeability property of the underlying signature scheme and it requires no extra assumptions.

Our approach can also be applied to some signature schemes with special properties, such as undeniable signatures, to obtain forward-secure signatures that still enjoy the added special property.

*Work done during September 1999; published in 7th ACM Conference on Computer and Communications Security, Nov. 2000.

†Department of Electrical Engineering, Technion, Haifa, Israel. hugo@ee.technion.ac.il

1 Introduction

A natural concern related to digital signatures is that the discovery of the signing key by an attacker provides this attacker with the power to forge the signature on any message. Not only this compromises the security and validity of any signature issued after the break but it also compromises *all* past signatures. Mechanisms such as key revocation (e.g., via certificate revocation) provide some protection against forgeries of signatures dated after the key's exposure is discovered, but are of no help to maintain the security of past signatures. One solution to this problem is the use of a (trusted) timestamping service applied to the signature string to validate its date of creation (e.g., [14]). Another solution is to change the pair of public-private keys related to the signature algorithm very often. A simpler approach to solve this problem was suggested by Ross Anderson [1] who called for finding signature schemes where signature keys expire periodically yet the public key does not change.

In a recent paper, Bellare and Miner [3] address this problem by formalizing Anderson's proposal through the notion of *forward-secure signatures*, and by providing the first efficient implementation of this notion. In their model time is divided into discrete periods (say, days, weeks, etc.). As in a regular digital signature scheme there is a public verification key and a secret signature key. However, while the public key does not change over time the secret key changes at each new period. The recipient of a signature can verify the signature against two parameters: correct verification by the public key and correspondence of the signature to a particular period of time. Such a signature scheme is called forward-secure if the exposure of the secret key at time-period t does <u>not</u> allow for forgery of signatures that belong to previous time-periods. Thus, in addition to the assurance given by regular digital signatures that the message was signed by the owner of the signing key, forward-secure signatures also provide a *proof of when* (i.e. in which period) the signature was issued.

There are many considerations regarding the design of forward-secure signature schemes including their motivation, usage, efficiency parameters, and the suitability of some simple approaches to solve the problem. We omit a detailed discussion of these issues here as they are thoroughly presented in [3] (the interested reader is highly encouraged to consult that paper). We focus in presenting our simple basic solution, its security, and variants.

As pointed out by [3] it is easy to build forward-secure schemes where one of the following basic parameters of the scheme grows linearly with *the total number of periods* T: the size of the secret key, the size of the public key and/or the size of the signature. They also point out to a "tree scheme" that requires $O(k \log T)$ size, where k is a security parameter (e.g., the size of the signature keys). However, it is argued in [3] that for a forward-secure signature scheme to be truly practical the size of these parameters should be independent of the number T of periods. (It seems that a minimal dependency with T is required if one wants to include an index to the time period in which a signature is generated – such information is essential to allow a secure binding between a signature and the period in which it was created.) Then [3] present the first solution to satisfy this size requirement. Their solution is based on a Fiat-Shamir-like signature scheme derived from identification schemes. Security is carefully formalized and proven based on the idealized random-oracle assumption.

Among the simple schemes pointed out by [3] there is one suggested by Ross Anderson [1] which can be applied to any signature scheme but which requires a secret key which is T times longer than the secret key of the underlying scheme. Here we show how to implement such a scheme with a *fixed-size secret key* while preserving the simplicity and generality of Anderson's solution. We use for that a pseudorandom generation procedure which by itself satisfies forward-security [4]. As a result our scheme enjoys some very desirable properties and some significant advantages over the solution of [3]:

Simplicity: our scheme is very simple and intuitive, and technically straightforward. This constitutes a conceptual and practical advantage as reflected in the following benefits.

Generality: any secure signature scheme can be made forward-secure using our approach. This means that we do not require the design and analysis of new digital signature schemes specially tailored to enjoy the forward-security property. Instead we can use any existing believed-to-be-secure signature scheme. In particular, we provide forward-secure RSA and DSS signatures (coming up with such schemes was left as an open problem in [3]).

Analysis: our construction does not necessitate of ideal assumptions such as random oracles. Security is proven, in a straightforward manner, on the sole basis of a secure regular digital signature scheme and the existence of pseudorandom functions (the existence of the latter is implied by the signature scheme).

Efficiency: the cost of signing in our solution is identical to the cost of signing under the underlying signature scheme. Verification requires two regular verifications (for verification of the message signature and a certificate).

Size of main parameters: Our scheme satisfies the requirement, postulated in [3], that the size of the main parameters (secret key, public key, signature) be independent of the total number of periods T for which the scheme is implemented. Both the secret and public key in our scheme are of size identical to the size of the corresponding keys in the underlying signature scheme; the signature's size is twice the size of a regular signature. In particular, using suitable signature schemes (e.g. DSS) the secret key in our scheme will be considerably shorter than the key in the scheme of [3]. The only dependency on T of the above parameters is the inclusion in each signature of an index to the period in which the signature was issued.

The one aspect in which our schemes are less efficient than the particular solution of [3] is the size of the memory maintained by the signer: in our case the signer keeps in memory T "certificates" one for each time period. This is usually not a real problem given that this information has no secrecy requirement neither there is a need to publish it or share it with others. Moreover, these certificates can be freely backed up by the signer to ensure availability (upon retrieval from the backup the signer can easily check the integrity of the certificates using its own public key); even if a particular certificate is eventually lost it will only prevent the signer from generating signatures for that period but will otherwise have no security effect (in particular, no compromise of any signature). If in some particular setting one wants to save in this memory requirement we show how to achieve that (see Section 3) at the expense of increased computation time and larger (by a $\log T$ factor) signature size.

Applicability to rich *signature schemes*: in addition to providing forward-security to regular digital signature schemes, our scheme can provide this feature to signature schemes with added capabilities such as undeniable signatures. We also discuss the design of forward-secure *threshold and proactive* signature schemes.

2 Forward-secure signatures from any signature scheme

2.1 The components of forward-secure signature schemes

We start with a regular digital signature scheme denoted SIG (e.g., RSA or DSS) composed of three algorithms: KG (the key generation algorithm), SIGN (the signature algorithm), and VER (the verification algorithm). Algorithm KG on input a security parameter and a random string outputs a pair of public and private keys denoted PK_0 and SK_0, respectively. Algorithm SIGN generates a signature on input a message M using the private key SK_0. Algorithm VER uses the public key PK_0 and is applied to pairs M, σ, its output is one of the values VALID or FAIL. It is required that VER returns VALID if and only if σ is a possible signature of M under SK_0.

Our goal is to design a forward-secure signature scheme based on SIG where the secret keys evolve over T periods of time, for some pre-specified number T and with each period being of some specified duration (a day, a week, etc.). We assume that the public key of the forward-secure scheme is to be used for a total time that does not exceed T periods (namely, after such T periods the public key is expired). On the other hand, the secret signing key changes at the beginning of each new period. A core security requirement is that exposure of the current secret key does not help the attacker in forging signatures from previous periods. We denote the resultant forward-secure signature scheme by FWSIG. Following the formalization of [3] we identify four components of FWSIG: FWKG, FWUPD, FWSIGN, FWVER, whose functionality we describe next.

The system (or user) initially generates secret and public parameters according to a key generation procedure FWKG (using a specified security parameter); this includes the generation of the public verification key PK_0 (called the *base* public key) that will remain fixed for the lifetime of the system (i.e., for T periods). The signing algorithm FWSIGN will depend on a secret key that will change with each time period. The signing key for period t is denoted by SK_t and will be generated at the beginning of each period out of information existent in the previous period via an *update* procedure denoted FWUPD. Given a message M to be signed, algorithm FWSIGN creates a signature on M, using key SK_t (and possibly other public information). Finally, algorithm FWVER is used to validate signatures. Its inputs consist of the base public key PK_0, a message M, a period number t, and a signature string s. It outputs VALID if and only if the signature s is a legal output during period t of algorithm FWSIGN applied to the message M (i.e., iff the signature was generated under SK_t).

We note that the verification algorithm in the forward-secure setting is verifying not only that the message M was signed by the "owner" of public key PK_0 but that it was specifically signed during time period t. It is the capability of verifying this fact that provides the added strength of forward-secure signature schemes relative to regular digital signature schemes.

We will denote a forward-secure signature scheme by the quadruple FWSIG = (FWKG, FWUPD, FWSIGN, FWVER) with the values T, the period duration, and security parameter being implicit parameters. Before moving to describe our implementation of these components (based on any regular signature scheme SIG) we introduce the following technical tool.

2.2 A technical tool: forward-secure prg's

The key-refreshment (or key-evolving) paradigm of forward-secure signatures is useful for many other cryptographic primitives with different security implications depending on the application. Examples include proactive systems (e.g., [16, 6, 15]) and key exchange protocols with key expiration and the related notion of "perfect forward-secrecy" [9]. In our construction of forward-secure signatures we use *forward-secure pseudorandom generators*. Such generators have been used in different contexts, e.g. [2, 6], and have simple realizations based on regular pseudorandom generators or pseudorandom functions. A formalization of this notion can be found in [4]. Here we describe them informally and point to one simple (generic) construction (other implementations are possible).

A forward-secure pseudorandom generator is one where seeds (or keys) are refreshed periodically and where exposure of the generator's secret state at a given time period reveals no (efficiently computable) information about the pseudorandom sequences generated in previous periods. Namely, the generator uses at each time period t a seed (or key) k_t to generate a sequence r_t which is indistinguishable from a truly random sequence as long as the keys $k_{t'}$ for $t' \leq t$ are not input to the distinguisher. In addition, the sequences r_t remain pseudorandom even if the distinguisher is given any key $k_{t'}$ for $t' > t$.

We present a simple construction of forward-secure pseudorandom generators based on any family $F = \{f_k\}$ of pseudorandom functions. For simplicity we assume that the output of the functions f_k is of the same length as the length of the index k; if a given pseudorandom family does not have this property then it can be achieved by simple output truncation (if the output is longer than the key) or by applying the function on different inputs (if the key is longer than a single output of the pseudorandom function). In each time period t we generate a pseudorandom sequence r_t, of a specified length, as follows. Let k_1 be a random index to a function in F. In period 1 we generate a pseudorandom sequence r_1 as $f_{k_1}(1), f_{k_1}(2), f_{k_1}(3), \ldots$ (the function is applied as many times as needed to reach the required length of r_1). At the beginning of each period $t \in \{2, \ldots, T\}$

we compute $k_t = f_{k_{t-1}}(0)$, and erase k_{t-1}. The sequence generated at this period is $r_t = f_{k_t}(1), f_{k_t}(2), f_{k_t}(3), \ldots$. It is easy to show that the exposure of k_t is of no help for distinguishing the sequences $r_{t'}$, $t' < t$, from randomness.

2.3 The general transformation

We now present our simple transformation of a regular signature scheme SIG into a forward-secure scheme FWSIG. We first outline the method and then provide a more detailed description. We start with a pair of secret-public keys for scheme SIG which we denote by SK_0 and PK_0, respectively. We then create T different secret keys, one for each time period, using a forward-secure pseudorandom generator out of an initial random seed k_0 and the key generation algorithm KG. These secret keys are generated as signature keys for the scheme SIG. The public verification keys corresponding to these signature keys are also generated and a "certificate" is created for each of them; that is, for each period $t = 1, \ldots, T$ we have a signature key SK_t, a corresponding public key PK_t, and a certificate CERT_t. Each certificate CERT_t includes the value PK_t, the period number t, and the value of the base public key PK_0 (it may also include additional information related to user U as well as other system parameters). Also included in the certificate is a signature under key SK_0 computed on the other information included in the certificate.

Once all this information is generated we erase all the secret keys (including SK_0), and all the information produced by the pseudorandom generator with the exception of the initial seed k_0 which we store and *keep secret*. We also keep all certificates and the public key PK_0. The public key PK_0 is treated as any other public key in a signature scheme, for example, it may be certified via a certification authority (in this case the certificate may include, in addition to standard certificate information, some specific information related to the forward-secure scheme such as the number of periods, their duration, etc.). The T certificates are also saved by U. There is no secrecy requirement on them, nor the need to make them public. They just need to be available to U during the corresponding time period. Changes to these certificates while stored can be detected (via the verification key PK_0). If a certificate is lost before the corresponding time period then U will not be able to generate any signature during that period but this will have no effect on the security of signatures generated in other periods. In any case guaranteeing the availability of these certificates is simple; in particular, because of the lack of secrecy requirements they can be freely backup-ed for availability.

At the beginning of each period t, the signature key SK_t is computed based on the key of previous period and the latter is then erased. Signatures during period t are generated using SK_t as the signature key; the corresponding certificate CERT_t is appended as part of the signature. Signature verification is done using the public key PK_t that appears in the certificate, while the certificate itself is validated using the base public key PK_0. The time period in which the signature was issued is verified via the period number that appears in the certificate.

A more careful and detailed description of our scheme follows.

UNDERLYING FUNCTIONS: We start with a regular digital signature scheme SIG = (KG, SIGN, VER), and a forward-secure pseudorandom generator FWPRG which on input k_{t-1} produces a pair of pseudorandom values k_t and r_t. (It suffices that each of k_t and r_t are individually pseudorandom; their joint distribution may not be pseudorandom, in particular, it can even be that $k_t = r_t$.)

INITIALIZATION (ALGORITHM FWKG): Here we describe the creation of parameters (secret and public) for the use of the scheme FWSIG by a user (we call it U), including the generation of the pair of verification-signature keys and other values necessary for the operation of the scheme. We assume the public verification key will be in use for T periods of time, each period being of some pre-specified length (a week, month, etc.). The following steps are performed by user U before the start of period 1.

1. Given a security parameter κ choose a random value r and compute $(PK_0, SK_0) \leftarrow \mathrm{KG}(\kappa, r)$.
2. Choose a random seed k_0 for FWPRG.
   For $t = 1$ to T do

   $(k_t, r_t) \leftarrow \mathrm{FWPRG}(k_{t-1})$

   $(SK_t, PK_t) \leftarrow \mathrm{KG}(\kappa, r_t)$

   $\mathrm{CERT}_t \leftarrow (PK_0, t, PK_t, \mathrm{SIGN}_{SK_0}(PK_0, t, PK_t))$
3. Erase SK_0 and k_t, r_t, SK_t, for $t = 1, \ldots, T$.
4. Store securely (i.e., in secret storage) the value k_0
5. Store CERT_t, $t = 1, \ldots, T$ and publish the public key PK_0 (e.g., via a certification authority).

UPDATE ALGORITHM (FWUPD): At the beginning of each period t do the following:

1. $(k_t, r_t) \leftarrow \mathrm{FWPRG}(k_{t-1})$
2. $(SK_t, PK_t) \leftarrow \mathrm{KG}(\kappa, r_t)$
3. retrieve CERT_t and verify that the values PK_0 and t in it are correct (i.e correspond to the base public key and current time period t); also check that the public key PK_t in it equals the public key generated in previous step. If any of these checks fail, abort.
4. store secretly k_t and SK_t and erase k_{t-1}

SIGNATURE ALGORITHM (FWSIGN): On input message M to be signed do:

1. Retrieve current values of CERT_t and SK_t.
2. Output the signature pair (CERT_t, σ) where $\sigma = \text{SIGN}_{SK_t}(M)$.

SIGNATURE VERIFICATION ALGORITHM (FWVER): On inputs the public key PK_0, a message M, a time period t, and a signature string s the verification algorithm FWVER proceeds as follows:

1. Parse s into the values CERT and σ.
2. Parse CERT to get the values $(PK_0', t', PK_t', \sigma')$.
3. Check that $PK_0' = PK_0$ and $t' = t$.
4. Verify that $\text{VER}_{PK_0}((PK_0, t, PK_t'), \sigma') = \text{VALID}$.
5. Verify that $\text{VER}_{PK_t'}(\sigma) = \text{VALID}$.
6. If *all* checks succeed output VALID, otherwise output FAIL.

2.4 Main Theorem

The following theorem summarizes our result and is straightforward to prove. The notion of unforgeability that we use for the regular underlying scheme is the strong notion of security for digital signature as formalized in [13] (security against existential forgery under adaptive chosen message attack). This notion can be extended in a natural way to capture also forward-security of signatures; this extension can be found in [3]. We omit the technical formalization details here.

Theorem 1 *Let* SIG = (KG, SIGN, VER) *be an unforgeable signature scheme and* FWPRG *be a forward-secure pseudorandom generator, then the scheme* FWSIG = (FWKG*,* FWUPD*,* FWSIGN*,* FWVER) *constructed above is an unforgeable forward-secure signature scheme.*

To prove the theorem one assumes the security of the forward-secure pseudorandom generator FWPRG and the unforgeability of the underlying signature scheme SIG. Then one shows that if a forger for the scheme FWSIG exist then one can construct out of it a forger for the scheme SIG, thus reaching a contradiction. We note that a basic difference between a forger against SIG and the one against FWSIG is that the former is never given the signature key, while the latter is provided with the signature key for a period t and it is considered successful if it finds a forgery for a signature corresponding to a period $t' < t$. See Appendix A for a proof of Theorem 1.

We end the section with a short analysis of the scheme's main parameters size.

SIZE OF MAIN PARAMETERS. The public key of our forward-secure scheme is a single regular public key (PK_0) for the underlying signature scheme SIG. The secret information maintained by the system, at any given period, are the period's signing key (corresponding to the underlying scheme SIG) and the current seed for FWPRG. (For applications where savings in secret storage is of prime importance, one can slightly modify the update scheme described above so that only the seed needs to be stored secretly while the signature key is re-computed upon need. Moreover, in some cases these two values may even be the same.) A signature string under our scheme includes a regular signature string under SIG and the period's certificate which includes the period number. (Note that for parties that verify multiple signatures for the same period, a single copy of the period's certificate suffices.)

3 Examples and Variants

DSS AND RSA. Building a forward-secure signature scheme based on DSS is very simple. The signature key in DSS is a random 160-bit quantity (taken modulo q). If one uses a forward-secure pseudorandom generator with seed of the same size (e.g., based on SHA-1) then the signature key and seed for a given period can be the same.[1]

In the case of RSA, the key generation procedure will use the value r_t produced in period t as input to a probabilistic algorithm that finds prime numbers. When the key is re-computed at the beginning of period t we will eventually find the same primes. We stress that one can use some simple optimizations that will save prime re-computation time during the update phase of period t. (For example, if one uses r_t as the key to a pseudorandom function and the tested primes are $f_{r_t}(1), f_{r_t}(2), \ldots$ then one can store the value of the inputs to the function where the chosen primes were found.) An RSA-based scheme where the periodic secret key is a uniformly chosen number (rather than a pair of prime numbers) is proposed next.

AN RSA-BASED VARIANT. As a curiosity, and maybe as a basis for future forward-secure schemes we outline the following scheme which is based on RSA-security. Its advantage over regular RSA is that the per-period keys are just random numbers (exponents) rather than (harder to generate) prime numbers. However, beyond being non-standard, it requires the use of a random-oracle (for non-interactive verification).

The key generation algorithm FWKG proceeds as follows. The base keys PK_0, SK_0 are regular RSA keys. That is, a pair of primes p, q is generated and the public key PK_0 is set to be $n = pq$ together with a public exponent e. (There are no more prime numbers generated in this scheme.) In addition, a fixed value $w \in Z_n^*$ is included as part of the

[1]Note, however, that there is an advantage to the case where a seed cannot be derived from a signature key: if some period's signature key gets exposed – e.g., via a physical attack – we would still like to keep the period's seed secret, or otherwise *all* subsequent periods are compromised. Note that the signature key may be more vulnerable than the seed given that it needs to be accessible during the whole period while the seed is used only during the update phase.

scheme's public key. Per-period keys are produced as follows. The secret key for period t is a random number $d_t < n$; these numbers are produced for all periods out of an initial seed using a forward-secure pseudorandom generator FWPRG (as described in our general scheme; in the present case, a period's seed and signature key can be the same). The public key for period t is the value $w^{d_t} \bmod n$. Once these public keys are generated, they are certified using RSA signatures with keys PK_0, SK_0; the key SK_0 is then erased (including the primes p, q and the signature exponent d).

A signature during period t on message m is produced as the string $S_m = m^{d_t} \bmod n$ together with a non-interactive proof that the discrete-log of S_m to the basis m equals the discrete-log of $w^{d_t} \bmod n$ to the basis w. (Recall that $w^{d_t} \bmod n$ is the certified public key for period t.) There are known efficient zero-knowledge interactive proofs for claims of this type which can be transformed into non-interactive proofs via the use of a "random-oracle" (a la Fiat-Shamir).

The security analysis of such a scheme can be based on the results of [12] (where a related scheme is used for undeniable signatures). In particular, that paper describes the relevant (interactive) zero-knowledge proofs as well as the way one has to choose the scheme's parameters (e.g., how to choose the primes p, q, which need be safe primes, or the value w).

HYBRID SCHEMES. In the general description of our schemes in Section 2 we assumed that the same signature scheme is used in producing per-period signatures as well as for certification signatures. We remark that this must not be the case and the two signatures can use different underlying schemes. Or they can use the same underlying scheme (say RSA) but with different security parameters. The previous RSA-based example is also an example of such an hybrid scheme.

SAVING CERTIFICATE SPACE. As pointed out before, the only parameter in our scheme that grows linearly with T is the amount of memory required by the signer for long-term storage of per-period certificates. As said, this need for (non-secret) storage will be seldom a practical problem. For completeness, however, we sketch a method for saving in the amount of required storage at the expense of increased signature computation time and increased ($\log T$) signature size. As before, the public key and required secret storage are of size *independent* from T. We keep most of our general scheme unchanged except that we change the way in which we use the base signing key to sign the per-period certificates. This change will dispense of the need for long-term storage for all the per-period certificates.

The idea is to build a (binary) Merkle's certification tree where the leaves are the per-period certificate information (this includes the period's public key and period's number but not a signature), while other nodes in the tree store a (collision-resistant) hash value computed on the concatenation of the values stored in its children nodes. The only use of the base signature key SK_0 is to sign the hash value in the tree's root. We call the resultant signature S. The signature on a per-period certificate will consist of this signature S together with a list of the ($\log T$) hash values corresponding to a path in the tree between the root and the corresponding certificate's leaf. However, instead of storing all this information during the T periods we will re-compute parts of it at the beginning of each period and derive from it the signature on the current period's certificate. Therefore, after producing the tree and signature S during the initialization phase we erase the tree and the signing key SK_0. We do keep the signature S.

Later, at each period's update phase, the signer re-computes all certificates for current and future periods (all information is derived from the current period's seed) and builds a partial hash tree (this partial tree only includes present and future certificate information). From this tree the signer derives all the hash values needed to be included (together with S) in the signature of the current period's certificate. Once this list of hashes is generated it is stored for the rest of the period (as part of the signature on the current certificate) and the rest of the tree is erased. (To be precise, another set of $\log T$ hashes in the tree, that are not reconstructible in the next period, need to be stored too; we omit the details.)

Thus we have saved the need for (long-term) storage of size proportional to T, but now the certificate's signature is longer (it includes $\log T$ hashes) and the update phase is more time consuming. Public and secret keys remain short as before.

UNDENIABLE SIGNATURES. The simplicity of our transformation of a digital signature scheme into a forward-secure one preserves many of the properties of the underlying signature scheme. A simple example of such a property is undeniable signatures [7]. It is easy to verify that our transformation applied to *any* undeniable signature scheme will result in *forward-secure undeniable signatures* (where the signature and verification procedures correspond to those used by the undeniable signatures scheme – some of which are carried as an interactive protocol). Similarly, there are other properties of signature schemes that are preserved by our transformation, thus resulting in forward-secure signatures that enjoy further qualities.

THRESHOLD AND PROACTIVE SIGNATURES. Here we discuss the combination of "threshold security" and forward-security for signature schemes. *Threshold signatures* [8, 10] are signature schemes in which the power to sign is distributed among several parties such that as long as less than a specified number (or threshold) of parties is corrupted, the signature scheme remains secure. The advantage of these schemes is in making the life of an attacker much harder; this attacker cannot find the key by just breaking into one location but has to be successful in its attempt in several locations. By adding the forward-security property to threshold signature one could achieve an even stronger security guarantee: even if at some point the attacker is able to break into a threshold of parties, the damage of signature forgery is confined to the period of time between key exposure and key revocation. That is, even such a successful attacker will *not* be able to endanger signatures corresponding to time-periods prior to the key exposure. A discussion on the design of forward-secure threshold schemes, including forward-secure proactive schemes, is presented in Appendix B.

Acknowledgment

This Research was supported by the Fund for the Promotion of Research at the Technion, and by Irwin and Bethea Green & Detroit Chapter Career Development Chair.

A Security of our forward-secure schemes

For completeness we outline here the proof of Theorem 1.

Let FW be a forger against scheme FWSIG that succeeds with probability ε. We build a forger F against the underlying scheme SIG as follows. Let (p, s) be a pair of public/private keys for SIG against which we want to produce forgeries. Forger F is given an oracle O_p that given a message returns a signature on that message under the pair (p, s). Forger F starts by generating information corresponding to T periods of a (modified) FWSIG scheme. F first chooses a period number t_0 at random between 1 and T. Then it chooses a random seed for FWPRG and generates out of it $T - t_0$ pairs of public/private SIG keys following the specification of the initialization algorithm FWKG. These pairs are set as the FWSIG keys for periods $t_0 + 1, t_0 + 2, \ldots, T$. In addition, F generates $t_0 - 1$ random and independent pairs of public/private keys that it sets as the FWSIG keys for periods 1 to $t_0 - 1$. For period t_0 it sets the public key to be p. Now F chooses a pair of public/private SIG keys (p', s') and produce certificates signed under s' for all the per-period public keys produced above. The public key p' becomes the base public key of scheme FWSIG.

Algorithm F now runs the forger FW against the (modified) FWSIG scheme defined above. We let FW query for signatures corresponding to any period of its choice except for the following restriction. Whenever FW asks for a signature corresponding to a period i, it cannot later ask for a signature corresponding to a previous period. Each time FW requests a signature (on a message of its choice) corresponding to any period different than t_0 then F provides the requested signature using its knowledge of the signature keys for those periods (these keys were chosen by F!) When FW asks to issue signatures for period t_0, then F goes to its oracle O_p to get the corresponding signatures under (p, s). When FW decides to query the secret information for some t'-th period then F does the following. If $t' \leq t_0$ then it aborts its run (i.e., in this case F fails to forge). If $t' > t_0$ then F provides FW with the secret information for that period (F knows it). F keeps running FW as before and responds to signature requests as before. If at some point FW outputs a forgery against a period $t'' < t'$ then F acts as follows. If $t'' \neq t_0$, F aborts its run failing to forge. Otherwise if $t'' = t_0$, F outputs the same forgery as FW did and stops. (Note that in order for FW's output to be considered a forgery it must be that FW did not ask for the forged message to be signed during period t_0, so in particular F did not ask for that signature from O_p meaning that this is a valid forgery for F too.)

What is the probability of F to succeed in forging? If FW succeeds with probability ε then F succeeds at least with probability roughly ε/T. This argument is outlined as follows.

First, the view of (the modified) FWSIG that F produces for FW is computationally indistinguishable from the view of FW under a real run of FWSIG (where all keys are produced out of a single initial seed for FWPRG). Indeed, using standard techniques it is straightforward to show that if a distinguisher exists for these two views of FW then we can construct a distinguisher for FWPRG. Next, conditioned on F choosing the value of t_0 as the period for which FW will eventually output a forgery, we have that the probability that F outputs a forgery against (s, p) is the same probability that FW succeeds in forging, i.e., probability ε. Since choosing the "right" t_0 happens with probability $1/T$ we get that ε/T is an approximate lower bound on the forging probability of F. (The "approximate" comes from the negligible probability with which the above mentioned views of FW can be succesfully distinguished.)

B On forward-secure threshold and proactive schemes

It is easy to design forward-secure threshold signatures if one is willing to maintain, at each participant, an amount of secret information that grows linearly with the number of time periods T. We start by sketching such a scheme and later show how to improve its efficiency. For concretenes, we assume a threshold signature scheme with a distributed key generation (DKG) protocol [17, 5, 11] (centralized key generation schemes can be used too but are less attractive). We assume n parties running the threshold scheme: $P_1, \ldots, P_n$.

1. Using protocol DKG the parties jointly generate base private and public signature keys SK_0, PK_0. The result of this computation is that each player P_i holds a share x_i of the base signature key while the base public key is known to all players. The latter is then published as the base public key of the forward-secure threshold signature scheme.

2. The joint key generation procedure DKG is repeated T times among the n players to produce secret shares (denoted $x_i(t), i = 1, \ldots, n, t = 1, \ldots, T$) for T different signature keys (these will act as the per-period signature keys) and the corresponding public keys. Per-period certificates for these public keys are produced and signed, jointly by the players (using shares x_i), under the base signature key SK_0. Each player P_i erases the share x_i that corresponds to the base signature key SK_0, and stores the T secret shares $x_i(t), t = 1, \ldots, T$.

3. During period t signatures are jointly produced by the players using the corresponding shares $x_i(t)$. These shares are *erased* at the end of the period.

It is easy to see that this process achieves both the threshold property and forward-security.

We show how to relax the requirement that each party stores T shares by using a forward-secure pseudorandom generator FWPRG. However, *this solution will only work* *against eavesdroppers*[2] (i.e., the attacker is allowed to learn all information in the memory of a corrupted party but is not allowed to change the behavior of that party). Step 1 above is not changed. In Step 2, each player P_i starts with a random seed s_i for FWPRG. From this seed, P_i derives T forward-secure pseudorandom values $s_i(t), t = 1, \ldots, T$, that he uses as the random input required for each of the T runs of DKG. Per-period certificates are produced as before. However, not only shares of the base signature key are erased as before but also other secret information (including the pseudorandom values $s_i(t)$ and shares $x_i(t)$) leaving only the seed s_i to be held secretly by player P_i, $i = 1, \ldots, T$. Then, at the beginning of each period a player P_i computes the new state $s_i(t)$ for FWPRG and erases the previous state. The n players then run DKG to (re)produce the t-th period shares $x_i(t)$.[3] These shares are used during the period to jointly sign messages.

An advanced variant of threshold signatures is the so-called *proactive signature schemes* [15]. In this context, time is partitioned into time periods (as with forward security) and a *mobile* attacker is considered which may control a party during some time periods but leave the party uncorrupted during other periods. In proactive threshold schemes, the sharing of the secret signature key is refreshed periodically, in such a way that the attacker now needs to break into a threshold of parties during a *single* time period (such as a day or week) before it can forge signatures.

One can see that the first straightforward solution to forward-secure threshold signatures presented above (where a party stores all shares for future periods) can be proactivized. This requires that at the beginning of each time period, when we perform the update operations required for forward-security, we also perform share refreshment operations for *all* the shares stored in a player's memory (i.e., the shares for all pending periods are refreshed). While the efficiency of this approach can be questioned, it does satisfy the properties of proactive forward-secure signature schemes.

Finally, we consider the proactivization of our second solution to forward-secure threshold schemes (the one involving a forward-secure pseudorandom generator). In this case, refreshing future shares as before seems hard, if at all possible. Indeed, when the attacker corrupts a party, it learns the current state of FWPRG as held by that party and then it learns all future shares; also those to be used after the attacker leaves the party. Yet, there is a possible direction to provide proactiveness in this setting. The idea is that the pseudorandom states $s_i(t)$ as used in the above second solution will not be generated using a forward-secure pseudorandom generator (as discussed in this paper) but rather using a *proactive pseudorandom protocol* as described in [6]. Assuming an eavesdropper-only attacker, this protocol constitutes a distributed pseudorandom generation procedure that is immune to mobile break-ins, and which provides each player with a pseudorandom value which is unpredictable for the adversary (except, of course, for values provided to players

[2]This severe restriction on the actions of the attacker is required during initialization and update phases; at other times both crash and malicious faults are also allowed.

[3]Note that if we had let corrupted players to deviate from their normal behavior during runs of DKG we could have ended with shares that are different than those created during Step 2.

currently controlled by the attacker). It is easy to see that replacing FWPRG in the above forward-secure threshold solution with such a proactive pseudorandom protocol results in a forward-secure proactive signature scheme.

References

[1] Anderson, R., Invited lecture, *Fourth Annual Conference on Computer and Communications Security*, ACM, 1997.

[2] Beaver, D., and Haber, S., "Cryptographic protocols provably secure against dynamic adversaries", *Eurocrypt '92*, LNCS No. 658, pages 307–323.

[3] Bellare, M., and Miner, S., "A Forward-Secure Digital Signature Scheme", *Advances in Cryptology – CRYPTO 99 Proceedings*, Lecture Notes in Computer Science, Vol. 1666, Springer-Verlag, M. Wiener, ed, 1999, pp. 431-438. Full version: Theory of Cryptography Library: Record 99-16, September 1999, `http://philby.ucsd.edu/cryptolib.html`.

[4] Bellare, M., and Yee, B., "Design and Application of Pseudorandom Number Generators with Forward Security", manuscript.

[5] Boneh, D., and Franklin, M., "Efficient generation of shared RSA keys", *Advances in Cryptology – CRYPTO 97 Proceedings*, Lecture Notes in Computer Science, Springer-Verlag Vol. 1294, B. Kaliski, ed, 1997, pp. 425–439.

[6] Canetti, R., and Herzberg, A., "Maintaining Security in the Presence of Transient Faults", *Advances in Cryptology – CRYPTO 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Springer-Verlag, Y. G. Desmedt, ed, 1994, pp. 425-438.

[7] Chaum, D. and Van Antwerpen, H., "Undeniable signatures", *Advances in Cryptology — Crypto '89 Proceedings*, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, G. Brassard, ed., 1989, pp. 212–217.

[8] Desmedt, Y. and Frankel, Y., "Threshold cryptosystems", *Advances in Cryptology — Crypto '89 Proceedings*, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, G. Brassard, ed., 1989, pp. 307–315.

[9] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", *Designs, Codes and Cryptography*, 2, 1992, pp. 107–125.

[10] Gennaro, R., Jarecki, S., Krawczyk H., and Rabin, T., "Robust Threshold DSS Signatures", *Advances in Cryptology – EUROCRYPT 96 Proceedings*, Lecture Notes in Computer Science Vol. 1070, Springer-Verlag, U. Maurer, ed, 1995, pp. 354–371.

[11] Gennaro, R., Jarecki, S., Krawczyk H., and Rabin, T., "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems", *Advances in Cryptology – EUROCRYPT 99 Proceedings*, Lecture Notes in Computer Science Vol. 1592, Springer-Verlag, J. Stern, ed, 1999, pp. 293–308.

[12] Gennaro, R., Krawczyk H., and Rabin, T., "RSA-based Undeniable Signatures", *Advances in Cryptology – CRYPTO 97 Proceedings*, Lecture Notes in Computer Science, Springer-Verlag Vol. 1294, B. Kaliski, ed, 1997, pp. 132-149.

[13] Goldwasser, S., Micali, S., and Rivest, R.L., "A digital signature scheme secure against adaptive chosen-message attacks", *SIAM J. Computing*, 17(2):281–308, April 1988.

[14] Haber, S. and Stornetta, W., "How to Time-Stamp a Digital Document", *Advances in Cryptology – CRYPTO 90 Proceedings*, Lecture Notes in Computer Science Vol. 537, Springer-Verlag, A. J. Menezes and S. Vanstone, ed., 1990.

[15] Herzberg A., Jakobsson, M., Jarecki, S., Krawczyk H., and Yung, M., "Proactive Public Key and Signature Systems", in *Proc. of the 4th ACM Conference on Computer and Communications Security*, 1997, pp. 100–110.

[16] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks", *Proc. of the 10th ACM Symposium on the Principles of Distributed Computing*, 1991, pp. 51-61.

[17] Pedersen, T., "A threshold cryptosystem without a trusted party", *Advances in Cryptology — Eurocrypt '91*, LNCS No. 547, pages 522–526.