A survey of key evolving cryptosystems

Matt Franklin

Department of Computer Science,
University of California,
Davis, CA, USA
E-mail: franklin@cs.ucdavis.edu

Abstract: This paper presents a survey of key evolving cryptosystems in the public key setting, focusing on two main approaches: 'forward security' and 'intrusion resilience'. The essential feature of this design strategy is that the secret key changes over time, while the corresponding public key remains unchanged. Key evolving cryptosystems can limit the damage caused by an attacker who occasionally learns your secret key.

Keywords: public key cryptography; key evolution; forward security; intrusion resilience.

Reference to this paper should be made as follows: Franklin, M. (2006) 'A survey of key evolving cryptosystems', *Int. J. Security and Networks*, Vol. 1, Nos. 1/2, pp.46–53.

Biographical notes: Matt Franklin is a Professor in the Computer Science Department at the University of California, Davis. He has published widely on many aspects of cryptography and security. He received a PhD in Computer Science from Columbia University in 1994.

1 Introduction

The security of cryptosystems depends on keeping secret keys secret, but this is quite hard to achieve in the real world. Attackers have a wide range of methods for learning a secret key, including brute-force search, cryptanalysis, system penetration, side channel attacks and social engineering. In recent years, cryptographic researchers have considered how to design cryptosystems that anticipate the likelihood of key exposure. The idea is to try to limit the damage that can result even when a determined attacker learns your secrets from time to time.

This paper presents a survey of one important design strategy of this type: key evolving cryptosystems in the public key setting. The essential feature of this design strategy is that the secret key changes over time, while the corresponding public key remains unchanged. Note that this is somewhat similar to the behaviour of a proactive threshold cryptosystem (Herzberg et al., 1997; Ostrovsky and Yung, 1991), although without splitting the functionality of the secret key among multiple parties (Desmedt and Frankel, 1989).

The emphasis in this survey is on two approaches to key evolving cryptosystems in the public key setting: 'forward secure' and 'intrusion resilient' cryptosystems. We begin with a discussion of forward secure cryptosystems, treating both forward secure digital signature and forward secure public key encryption schemes. Then, we describe intrusion resilient cryptosystems, focusing on intrusion resilient public key encryption, and then touching briefly on constructions for intrusion resilient digital signatures. The final section gives some conclusions and further pointers to the literature.

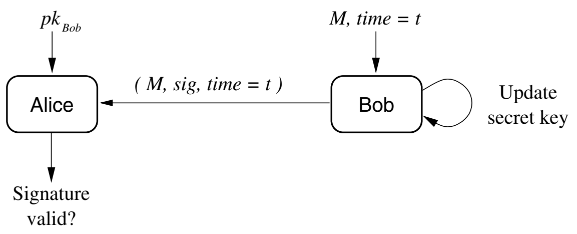
2 Forward security in the public key setting

In this section, we survey forward secure cryptosystems in the public key setting. This approach emerged from suggestions of Back (1996) and Anderson (1997), with the first formalisation by Bellare and Miner (1999). We begin with a discussion of forward secure digital signature schemes, followed by a treatment of forward secure public key encryption.

2.1 Forward secure signatures

A forward secure signature scheme is a public key signature scheme where the signing key changes over time. There is an initial signing key for the first time period. At the end of each time period, an efficient 'key update' function computes the new signing key for the next time period, and erases the old signing key. The verification function takes four inputs: the message, the signature, the public key and the time period. The signing function takes four inputs: the message, the public key, the time period and the signing key for the time period (see Figure 1).

Figure 1 Forward secure signature



The security guarantee is that exposure of the signing key for any time period does not enable forged signatures for earlier time periods. This is the best that is possible, because from any signing key, an attacker can compute the signing keys for all future time periods.

There are many constructions for forward secure signatures in the cryptographic literature, including Bellare and Miner (1999), Abdalla and Reyzin (2000), Itkis and Reyzin (2001), Malkin et al. (2002) and Kozlov and Reyzin (2002). See Cronin et al. (2003) for some interesting efficiency comparisons of various forward secure signature schemes. The relation of forward secure signatures to other kinds of key evolving signature primitives is explored by Malkin et al. (2004).

Here, we sketch two basic constructions of Malkin et al. (2002). Let T denote the maximum allowable number of time periods in a forward secure signature scheme. Note that any ordinary public key signature scheme can be viewed as a forward secure signature with $T = 1$ (by erasing the signing key at the end of the single time period).

2.1.1 Sum composition

Let Σ_0 and Σ_1 be forward secure signature schemes with maximum allowable number of time periods T_0 and T_1 respectively. The sum composition (denoted $\Sigma_0 \oplus \Sigma_1$) has maximum allowable number of time periods $T = T_0 + T_1$. Intuitively, Σ_0 is used to sign messages for the first T_0 time periods, and Σ_1 is used to sign messages for the last T_1 time periods.

We begin with a simplified variant. The public key consists of the public keys of both Σ_0 and Σ_1. The secret key starts with the secret keys for the first time period of both Σ_0 and Σ_1. At the end of time period i, where $i < T_0$, the Σ_0 signing key is updated to time period $i + 1$. At the end of time period T_0, the Σ_0 signing key is completely erased. At the end of time period $T_0 + i$, where $i < T_1$, the Σ_1 signing key is updated to time period $i + 1$. To verify a signature from time period i, where $i \leq T_0$, use the public key for Σ_0 with time period i. To verify a signature from time period $T_0 + i$, where $i \leq T_1$, use the public key for Σ_1 with time period i.

The actual sum composition includes some efficiency improvements. Firstly, the public key actually consists of just the hash of the public keys of Σ_0 and Σ_1. Then each signature during any time period includes the public keys for both Σ_0 and Σ_1, and these are checked against the public hash as part of the verification procedure.

The other efficiency improvement reduces the overall size of the secret key. Let r_1 be the random seed that is used by the key generation algorithm of Σ_1 to generate the public key and signing key for Σ_1. Then the secret key for $\Sigma_0 \oplus \Sigma_1$ starts with the following four values: the secret key of Σ_0 for its first time period, the random seed r_1 and the public keys for both Σ_0 and Σ_1. At the end of time period T_0, the secret key of Σ_1 for its first time period is recomputed from the seed r_1 (and the secret key for Σ_0 is deleted).

The sum composition can be iterated. For example, suppose that Σ_0 and Σ_1 are ordinary signature schemes. Then $T_0 = T_1 = 1$, and the 'level one' composed scheme has maximum allowable number of time periods $T = 2$. If both Σ_0 and Σ_1 are level one sum compositions, then their 'level two' sum composition has maximum allowable number of time periods $T = 4$.

2.1.2 Product composition

Let Σ_0 and Σ_1 be forward secure signature schemes with maximum allowable number of time periods T_0 and T_1, respectively. The product composition (denoted $\Sigma_0 \otimes \Sigma_1$) has maximum allowable number of time periods $T = T_0 \times T_1$. Intuitively, there is a unique instance of Σ_1 for signing messages 'within' each time period of T_0. Each time period of Σ_0 is now called an 'epoch' because it actually lasts for T_1 time periods. Let (i, j) denote the jth time period of the ith epoch.

The public key of $\Sigma_0 \otimes \Sigma_1$ is the public key of Σ_0. A signature of a message in time period (i, j) contains the following values: the message signed with the jth secret key of the ith instance of Σ_1, and the public key of the ith instance of Σ_1 signed with the ith secret key of Σ_0. The secret key in time period (i, j) contains the ith secret key of Σ_0, the public key for the ith instance of Σ_1, the jth secret key for the ith instance of Σ_1 and a random seed for generating future public keys and private keys of Σ_1. The verification and update algorithms are omitted.

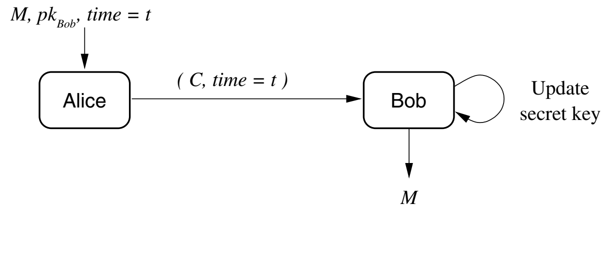
2.1.3 MMM construction

A hybrid variant of the sum and product compositions yields a particularly efficient forward secure signature scheme with essentially no restriction on the maximum allowable number of time periods. See Malkin et al. (2002) for details.

2.2 Forward secure encryption

Forward Secure Encryption (FSE) is a public key encryption scheme where the secret key changes over time. There is an initial secret key for the first time period. At the end of each time period, an efficient 'key update' function computes the new secret key for the next time period, and erases the old secret key. The encryption function takes three inputs: the message, the public key and the time period. The decryption function takes four inputs: the ciphertext, the public key, the time period and the secret key for the current time period (Figure 2).

Figure 2 Forward secure encryption



The security guarantee is that exposure of the secret key for any time period does not compromise any ciphertexts that were encrypted for earlier time periods. This is the best that is possible. Once the attacker knows one secret key, he/she could perform the same efficient key update function to compute the secret keys for all future time periods.

In this section, we describe a construction for forward secure encryption due to Canetti et al. (2003). We begin by giving a brief discussion of bilinear pairings for cryptography followed by a description of a building block called 'binary tree encryption'.

2.2.1 Bilinear pairings for cryptography

Let G_1 and G_2 be two groups of order q for some large prime q. It is standard to view G_1 as an additive group and G_2 as a multiplicative group. Let $\hat{e}$ be a function (or 'map') from $G_1 \times G_1$ to G_2. We say that $\hat{e}$ is 'bilinear' if $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in G_1$ and all integers a, b.

The Bilinear Diffie-Hellman (BDH) assumption is as follows. Let a, b, c be chosen randomly from $[1, \ldots, q-1]$, and let P be chosen randomly from G_1. Given (P, aP, bP, cP), it is hard to compute $\hat{e}(P, P)^{abc}$.

Actually, the BDH assumption is a conjecture about hardness as the size of the problem increases. To state the assumption more formally, we define a 'BDH parameter generator'. This is an efficient randomised algorithm that takes as input a 'security parameter' k, and outputs descriptions of groups G_1 and G_2 of prime order $q > 2^k$, together with the description of an efficiently computable bilinear map $\hat{e} : G_1 \times G_1 \to G_2$.

Now, let A be any randomised algorithm for solving BDH problem instances with respect to any $G_1, G_2, \hat{e}$ that might be output by the BDH parameter generator. The BDH assumption says that if the running time of A is polynomial in k, then the probability of success for A is asymptotically negligible (i.e. less than $1/k^{\lambda}$ for all constant λ when k is sufficiently large). Here, the probability of success for A is over the random choices of the BDH parameter generator, the random choice of P and the random choice of a, b, c.

A number of candidate constructions for BDH parameter generators have been proposed that are conjectured to satisfy the BDH assumption. In these constructions, the group G_1 may be a special kind of elliptic curve or algebraic variety, and the bilinear mapping may be the modified 'Weil pairing' or 'Tate pairing'. See Boneh and Franklin (2003) for a discussion of some of these constructions. Further details about the underlying mathematics will not be needed for this survey.

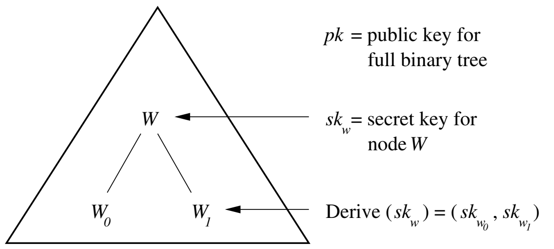
2.2.2 Binary tree encryption

Let T be a full binary tree of depth ℓ. Let each node in the tree be labelled by a bitstring as follows. The root is labelled with the empty string ϵ. The left child of the root is labelled by 0 and the right child of the root is labelled by 1. The four nodes at depth 2 have labels 00, 01, 10, 11 (from left to right). In general, if a node is labelled with bitstring w, then its left child has label w_0 and its right child has label w_1. Note that all of the nodes at depth t in the tree are labelled with bitstrings of length t.

There is a single public encryption key PK for the tree. There is a secret decryption key for each node of the tree. Let SK_w denote the secret key of the node with label w. The functionality of binary tree encryption and decryption is as follows (see Figure 3):

- Encryption takes three inputs: some message M, some node label w and the public key PK. It returns a ciphertext C.
- Decryption takes three inputs: some ciphertext C, some node label w and the secret key SK_w. It returns the original message M (assuming that the same node label was used for encryption).

Figure 3 Binary tree encryption


pk = public key for full binary tree
W
sk_w = secret key for node W
W_0
W_1
Derive $(sk_w) = (sk_{w_0}, sk_{w_1})$

In addition, there is an efficient 'key derivation' algorithm with the following functionality. It takes as input the public key PK, some node with label w, and the secret key SK_w. It outputs secret keys SK_{w0} and SK_{w1} for the nodes with labels w_0 and w_1.

The existence of an efficient key derivation algorithm implies that any ciphertext for the node labelled by w can be decrypted, given the secret key for any ancestor of the node in the tree. Intuitively, the security guarantee for binary tree encryption is that no other ciphertexts are compromised by the exposure of the secret key for any node in the tree (See Canetti et al. (2003) for details and variants of the security model).

For simplicity, the specification of binary tree encryption does not explicitly mention key generation.

2.2.2.1 Construction: Here is one construction for binary tree encryption, due to Canetti et al. (2003) and based on earlier constructions for identity-based encryption (Boneh and Franklin, 2003; Gentry and Silverberg, 2002).

Let $\hat{e}, G_1, G_2$ be a hard instance of the BDH problem, where G_1 and G_2 are groups of order q for some large prime q. Let P be a random element of G_1. Let $Q = aP$, where a is some random integer in $[0, \ldots, q-1]$. Let H_1 be a hash function from bitstrings of length at most ℓ to G_1. Let H_2 be a hash function from G_2 to bitstrings of length n, where n is the length of the messages to be encrypted. Let ℓ be the depth of the binary tree.

The public key for the binary tree will consist of q, ℓ, P, Q and descriptions of $\hat{e}, G_1, G_2, H_1, H_2$.

Let $w = w_1, w_2, \ldots, w_t$ be the label of a node at depth t in the tree. The secret key SK_w will consist of $t+1$ elements of G_1. These $t+1$ elements will be denoted $(R_{w|1}, R_{w|2}, \ldots, R_{w|t-1}, R_w, S_w)$, where $w|i$ denotes the first i bits of w. For example, the root has secret key $\mathrm{SK}_\epsilon = (S_\epsilon)$, the left child of the root has secret key $\mathrm{SK}_0 = (R_0, S_0)$ and the right child of the left child of the root has secret key $SK_{01} = (R_0, R_{01}, S_{01})$.

Here is a description of the key derivation algorithm, given the secret key SK_w for some node with label w. Choose random integers ρ_{w0} and ρ_{w1} in $[1, \ldots, q-1]$. Let $R_{w0} = \rho_{w0}P$ and $R_{w1} = \rho_{w1}P$. Let $S_{w0} = S_w + \rho_{w0}H_1(w0)$ and $S_{w1} = S_w + \rho_{w1}H_1(w1)$. Then the secret key for the left child is $SK_{w0} = (R_{w|1}, R_{w|2}, \ldots, R_{w|t-1}, R_w, R_{w0}, S_{w0})$ and for the right child is $SK_{w1} = (R_{w|1}, R_{w|2}, \ldots, R_{w|t-1}, R_w, R_{w1}, S_{w1})$. The root key SK_ϵ can be computed as $aH_1(\epsilon)$.

Consider encryption under the public key PK of some n-bit message M for some node with label w at depth t in the tree. The ciphertext always consists of $t+2$ items that can be viewed as the message exclusive-OR'ed with a random one-time pad, together with $t+1$ 'hints' to help the decryptor recover the pad. The encryptor chooses a random $\gamma \leftarrow [1, \ldots, q-1]$ and computes $(\gamma P, \gamma H_1(w_1), \gamma H_1(w_1w_2), \ldots, \gamma H_1(w), M \oplus H_2(\hat{e}(Q, H_1(\epsilon))^\gamma))$. In the special case of encryption for the root, the ciphertext includes just one hint γP.

To decrypt ciphertext $C = (U_0, U_1, \ldots, U_t, V)$ for the node with label w and secret key SK_w, proceed as follows. The decryptor uses the 'hints' to recover the random one-time pad. Let num $= \hat{e}(U_0, S_w)$. Let denom $= \prod_{i=1}^{t} \hat{e}(R_{w|i}, U_i)$. Then $d = H_2(\text{num}/\text{denom})$ is the random one-time pad used by the encryptor. That is, $M = V \oplus d$. In the special of decryption for the root, denom = 1 and $d =$ num. To recap:

- *Binary Tree Public Key:* $(q, G_1, G_2, \hat{e}, P, Q = \alpha P, H_1, H_2)$, for random $\alpha \leftarrow [1, \ldots, q-1]$.
- *Binary Tree Root Secret Key:* $\alpha H_1(\epsilon)$.
- *Binary Tree Key Derivation:* choose random $\rho_{w0}, \rho_{w1} \leftarrow [1, \ldots q-1]$, and then
  - $SK_{w0} = (R_{w|1}, R_{w|2}, \ldots, R_{w|t-1}, R_w, R_{w0} = \rho_{w0}P, S_{w0} = S_w + \rho_{w0}H_1(w0))$.
  - $SK_{w1} = (R_{w|1}, R_{w|2}, \ldots, R_{w|t-1}, R_w, R_{w1} = \rho_{w1}P, S_{w1} = S_w + \rho_{w1}H_1(w1))$.
- *Binary Tree Encrypt of M for node w:* $(\gamma P, \gamma H_1(w_1), \gamma H_1(w_1w_2), \ldots, \gamma H_1(w), M \oplus H_2(\hat{e}(Q, H_1(\epsilon))^\gamma))$.
- *Binary Tree Decrypt of $(U_0, U_1, \ldots, U_t, V)$ for node w:* $V \oplus (H_2(\hat{e}(U_0, S_w)/(\prod_{i=1}^{t} \hat{e}(R_{w|i}, U_i))$.

The correctness of this construction relies on the bilinearity of the mapping $\hat{e}$. To see that decryption is successful, note that the decryptor reconstructs the same random one-time pad as the encryptor:

$$\begin{aligned}\text{num} &= \hat{e}(U_0, S_w)\\ &= \hat{e}\left(\gamma P, \alpha H_1(\epsilon) + \sum_{i=1}^{t} \rho_{w|i} H_1(w|i)\right)\\ &= \hat{e}(P, H_1(\epsilon))^{\alpha\gamma} \left(\prod_{i=1}^{t} \hat{e}(P, H_1(w|i))^{\gamma\rho_{w|i}}\right)\end{aligned}$$

$$\begin{aligned}\text{denom} = \prod_{i=1}^{t} \hat{e}(R_{w|i}, U_i) &= \prod_{i=1}^{t} \hat{e}(\rho_{w|i}P, \gamma H_1(w|i))\\ &= \prod_{i=1}^{t} \hat{e}(P, H_1(w|i))^{\gamma\rho_{w|i}}\end{aligned}$$

Thus, num/denom $= \hat{e}(P, H_1(\epsilon))^{\alpha\gamma}$ as needed.

The security of this binary tree encryption scheme can be strengthened by using a variant of the Fujisaki–Okamoto transform (Fujisaki and Okamoto, 1999); see Canetti et al. (2003) for details. A recent improved construction for binary tree encryption is due to Boneh et al. (2005).

2.2.3 *FSE from binary tree encryption*

The CHK construction for FSE is built from binary tree encryption as follows. Let T be a complete binary tree of depth ℓ. Then T has $2^{\ell+1} - 1$ nodes. Let the nodes be labelled with bitstrings as described earlier, and assume that we are given a binary tree encryption scheme for T.

Each node of T will correspond to a unique time period, according to a 'preorder traversal' of T. A preorder traversal is the order in which nodes are *first visited* in a depth-first search of the tree. For convenience, the time periods are numbered from 0 to $N-1$, where $N = 2^{\ell+1} - 1$.

The public encryption key for the FSE scheme is the public key of the underlying binary tree encryption scheme. To encrypt a message during the ith time period in the FSE scheme, encrypt the message in the underlying binary tree encryption scheme with respect to the ith node in the preorder traversal of the tree.

The secret decryption key for the FSE scheme is a subset of secret decryption keys for the binary tree encryption scheme. The subset changes for each time period. The crucial invariant that must be maintained is that the key derivation algorithm for the binary tree encryption scheme, when applied repeatedly to the keys in the subset, should yield keys for the nodes corresponding to all time periods greater than or equal to i, but not for any time periods less than i.

The efficient key update function for the FSE scheme is a simple stack-based algorithm that maintains this invariant. The secret keys in the subset will be organised as a stack. During time period i, the secret key at the top of the stack will always be SK_w, where the node with label w is the ith node visited in the preorder traversal of the tree T. Initially, the stack contains only SK_ϵ.

Here are the details of the FSE construction:

- Construct a binary tree encryption scheme for a complete binary tree T of depth ℓ, with public key PK, secret key SK_w for node with label w, key derivation algorithm Der(PK, w, SK_w), encryption algorithm Enc(PK, w, M) and decryption algorithm Dec(PK, w, SK_w, C).
- *FSE Public Key:* PK (binary tree public key).
- *FSE Initial Secret Key:* SK_ϵ (binary tree root secret key).
- *FSE Encrypt message M in time period i:* Enc(PK, w, M), where the node with label w is the ith node visited in the preorder traversal of the tree T.

- *FSE Decrypt ciphertext C in time period i:* Dec(PK, w, SK_w, C), where the node with label w is the ith node visited in the preorder traversal of the tree T.
- *FSE key update at end of time period i:* pop the top key off of the stack. This is SK_w, where the node with label w is the ith node visited in the preorder traversal of the tree T. If w is not a leaf, then run the key derivation algorithm to compute SK_{w0} and SK_{w1}, and push them onto the stack in reverse order (first push SK_{w1} and then push SK_{w0}).

2.2.3.1 Example Suppose that the complete binary tree T has depth 3. A preorder traversal of T would visit its 15 nodes in this order: ϵ, 0, 00, 000, 001, 01, 010, 011, 1, 10, 100, 101, 11, 110, 111. The node labelled by ϵ (root) corresponds to time period 0, the node labelled by 0 corresponds to time period 1, the node labelled by 00 corresponds to time period 2 and so forth.

During time period 0, the subset of keys consists of just SK_ϵ. At the end of time period 0, pop SK_ϵ off the stack, derive $\mathrm{SK}_0, \mathrm{SK}_1$ and push them onto the stack in reverse order. During time period 1, the subset of keys consists of $\mathrm{SK}_0, \mathrm{SK}_1$ organised as a stack (in this order from top to bottom). At the end of time period 1, pop SK_0 off the stack, derive SK_{00}, SK_{01}, and push them onto the stack in reverse order. During time period 2, the subset of keys consists of $\mathrm{SK}_{00}, \mathrm{SK}_{01}, \mathrm{SK}_1$, organised as a stack (in this order from top to bottom). At the end of time period 3, pop SK_{00} off the stack, derive SK_{000}, SK_{001} and push them onto the stack in reverse order. During time period 4, the subset of keys consists of $\mathrm{SK}_{000}, \mathrm{SK}_{001}, \mathrm{SK}_{01}, \mathrm{SK}_1$ organised as a stack (in this order from top to bottom). At the end of time period 4, pop SK_{000} from the stack and do nothing else, because this corresponds to a leaf in the tree. During time period 5, the subset of keys consists of $\mathrm{SK}_{001}, \mathrm{SK}_{01}, \mathrm{SK}_1$. Note that the secret key for the current time period is always at the top of the stack. Also note that the keys in the stack for the current time period can be used to derive all future keys but no past keys.

Here is an alternative description of the stack of secret keys during the time period i, $i \geq 1$. Let w be the label of the ith node visited in the preorder traversal of the tree. Then SK_w is at the top of the stack. Let $w = w_1, \ldots, w_t$ be the binary expansion of w, where t is the depth of the node in the tree. Then the rest of the stack consists of keys of the form $\mathrm{SK}_{w_1,\ldots,w_{j-1}1}$, where $1 \leq j \leq t$ and $w_j = 0$. These are sorted by the size of j, with the smallest j on the bottom of the stack.

The security of the FSE scheme is closely related to the security of the underlying binary tree encryption scheme. In particular, it can be shown to resist a very strong chosen ciphertext attack when the underlying binary tree encryption scheme has been strengthened with the Fujisaki–Okamoto transformation. The proof of security is in the random oracle model, and under the BDH assumption.

Remarkably, Canetti et al. show that the random oracle model can be avoided here. That is, protection against a very strong chosen ciphertext attack can be proved for a different FSE construction under the related 'Decisional Bilinear Diffie-Hellman Assumption'. See Canetti et al. (2003) for details.

3 Intrusion resilience in the public key setting

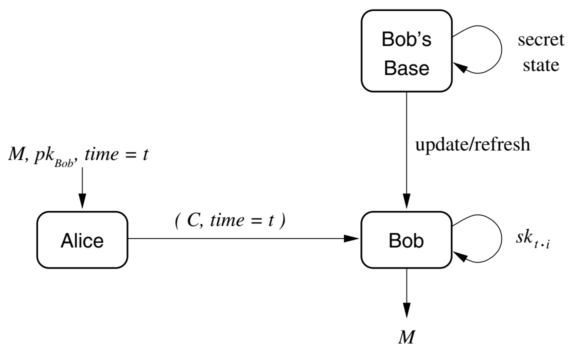
In this section, we survey intrusion resilient cryptosystems in the public key setting. We begin with intrusion resilient public key encryption, and then discuss intrusion resilient signature schemes.

The notion of intrusion resilience – an elaboration of forward security in the public key setting – was first formulated by Itkis and Reyzin (2002), building on ideas for 'key-insulated' cryposystems due to Dodis et al. (2002, 2003). The architecture for key-insulated cryptosystems is similar to that of intrusion resilient cryptosystems, but without incorporating forward security (and in fact allowing a kind of 'random access' of the secret keys).

3.1 Intrusion resilient encryption

Intrusion Resilient Encryption (IRE) (Dodis et al., 2003) is an extension of the functionality of forward secure encryption. There are two entities, called the 'user' and the 'base'. There is a single public encryption key that does not change over time. The user has a secret decryption key that changes over time. The base also has secret information that changes over time. In addition, there is a flow of secret messages in one direction only, from the base to the user. Thus, the evolution of the user's secrets is influenced by the secret messages sent to it by the base (see Figure 4).

Figure 4 Intrusion resilient encryption



There are actually two levels of granularity of time in an IRE scheme. At the upper level, time is divided into periods just as in forward secure encryption. These periods are numbered $0, 1, 2, \ldots, N-1$.

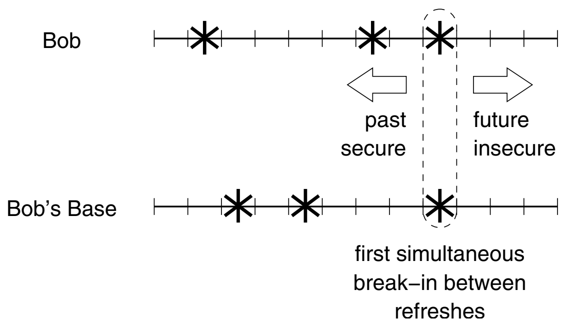
At the lower level, each time period is divided into an arbitrary number of 'refresh sub-periods'. The sub-periods of time period i are numbered $i0, i1, \ldots, ir$. The value of r may be different for each time period, and is not determined beforehand. For convenience, the first sub-period of each period i is numbered $i0$.

Only the upper level time granularity is visible to the outside user. Anyone who encrypts a message using the public key only needs to know the current time period, and not the current refresh sub-period. The lower level time granularity reflects the frequency with which the user and base re-randomise their secret information. The user's secret key may be different in every refresh sub-period of period i. Nevertheless, all of these refreshed versions of the user's secret key will be able to decrypt a message that was encrypted during period i.

The power of the lower level time granularity is that it imposes an extra restriction on an attacker. The attacker learns nothing useful unless he/she compromises the user and the base in the same refresh sub-period. The size of the refresh sub-period can be made very small to make the attacker's task difficult, while the upper level time granularity can be more leisurely to simplify the encryption process for outside users.

The security guarantees are very strong. If the secrets of the user and the base are compromised in the same refresh sub-period, then the system ensures 'forward security'. That is, the attacker gains the ability to decrypt all future ciphertexts, while all past ciphertexts remain unreadable. If the secrets of the user are compromised in any period, then of course the attacker gains the ability to decrypt all ciphertexts for that period. Otherwise, the user and base may be compromised repeatedly by the attacker over time with no loss of security whatsoever (see Figure 5). In particular, note that the adversary gains no advantage when the secrets of the base only are compromised in any time period.

Figure 5 Intrusion resilience security



3.1.1 *Functional specification of IRE*

An IRE scheme has the following components:

- An efficient encryption algorithm (Enc) that takes as input a public key PK, a time period i and a message M. The output is the ciphertext (i, C).
- An efficient decryption algorithm (Dec) that takes as input the public key PK, the current secret key SK_{ir} and the ciphertext (i, C). The output is the original message M. That is, decryption (during any refresh sub-period of time period i) inverts encryption (during time period i) in the standard way.
- An efficient base key update algorithm (UpdBase), that takes as input the current base key $\mathrm{SK}B_{ir}$. The output is a new base key $\mathrm{SK}B_{i+1.0}$ for the next time period, and a key update message $\mathrm{SK}U_i$ to be sent to the user.
- An efficient user key update algorithm (UpdUser), that takes as input the current user key $\mathrm{SK}U_{ir}$ and a key update message $\mathrm{SK}U_i$ sent from the base. The output is a new user key $\mathrm{SK}_{i+1.0}$ for the next time period.
- An efficient base key refresh algorithm (RefBase), that takes as input the current base key $\mathrm{SK}B_{ir}$. The output is a new base key $\mathrm{SK}B_{ir+1}$ (for the next refresh sub-period of the current time period) and a key refresh message $\mathrm{SK}R_{ir}$ to be sent to the user.
- An efficient user key refresh algorithm (RefUser), that takes as input the current user key SK_{ir} and a key refresh message $\mathrm{SK}R_{ir}$ from the base. The output is a new user key $\mathrm{SK}_{i.r+1}$ (for the next refresh sub-period of the current time interval).

For simplicity, the functional specification does not explicitly include the key generation algorithm.

3.1.2 *Construction for IRE*

The construction for IRE is closely related to the earlier construction for forward-secure encryption. The IRE construction is also built on top of a binary tree encryption scheme. Essentially the same binary tree encryption scheme is used, although a small optimisation is made in the key derivation algorithm.

In the IRE construction, only the leaves of the binary tree correspond to the time periods. Thus, a complete binary tree of depth ℓ can support $N = 2^\ell$ time periods. The time interval t corresponds to the leaf whose ℓ-bit label is a binary representation of t. The ℓ-bit binary representation of t is denoted $< t >= t_1, \dots, t_\ell$. For notational convenience, we sometimes view $< t >$ as the string $t_0, t_1, \dots, t_\ell$ where $t_0 = \epsilon$ (the empty string).

During any time period t, the user has the binary tree secret key for the leaf with label $< t >$. This will be sufficient for the user to decrypt all of the ciphertexts for that time period.

Let $\rho(t) = \{t_0, t_1, \dots, t_{j-1}1 : t_j = 0\}$. (For example, when $\ell = 4$, $\rho(2) = \{1, 01, 0011\}$.) The user and base will share the binary tree secret keys for all of the nodes whose labels are in $\rho(t)$. This set consists of all right siblings of the nodes on the path from the root to the leaf with the appropriate label. Notice that every leaf whose label is larger than t is contained in the subtree rooted at some node whose label is in $\rho(t)$.

During each time period t, the user holds shares of all keys for nodes whose labels are in $\rho(t)$, along with the secret key for the leaf with label $< t >$. The base holds the corresponding shares of all keys for nodes whose labels are in $\rho(t)$, but the base holds no information about the secret key for the leaf with label $< t >$. The user's information is sufficient to decrypt all messages for time period $< t >$, but insufficient to execute a key update by itself. The base's information is insufficient to decrypt messages for time period $< t >$ by itself, and insufficient to execute a key update by itself.

Together, however, the base and the user have sufficient information to perform a key update. In fact, the key update procedure only needs a single message from the base to the user. The key refresh procedure is also done with a single message from the base to the user (Dodis et al., 2003).

3.2 Intrusion resilient signatures

The notion of 'intrusion resilient signature' is analogous to the notion of IRE described in the preceding section. There are again two entities: user and base. The user' secret signing key changes over time, and the corresponding public verification key does not change. The base's secret information changes over time. The flow of secret messages, and the evolution of the user's secrets, is the same as with IRE. Figure 4 can be simply changed to capture the desired functionality, as follows: flip the arrow from *M* to Bob (input to Bob's signing algorithm), flip the arrow from Bob to Alice and change *C* to sig (output of Bob's signing algorithm) and add an arrow leading out of Alice ('yes/no' output of Alice's signature verification algorithm).

The two levels of granularity of time are the same as with IRE, and Figure 5 illustrates this with no changes needed. The security guarantees are similarly strong. Forward security for the signature scheme is preserved if user and base are compromised in the same refresh sub-period. Otherwise, the attacker only gains the ability to sign for those specific time periods during which he/she successfully compromises the user.

The first construction of an intrusion resilient signature scheme (Itkis and Reyzin, 2002) is based on a forward secure signature scheme by the same authors (Itkis and Reyzin, 2001). Its security is based on a variant of the strong RSA assumption (for RSA moduli restricted to products of safe primes), with a proof of security in the random oracle model. The algorithms for signing and verifying are quite efficient.

A generic construction for intrusion resilient signature schemes is given by Itkis (2002). It can be built from any ordinary signature scheme. This scheme is proved secure with respect to a somewhat stronger notion of security than (Itkis and Reyzin, 2002), that is, to withstand a 'fully adaptive' adversary. Intuitively, a fully adaptive adversary can defer each attack decision within a single round until the last possible moment.

4 Conclusion and further pointers to the literature

In conclusion, we have surveyed key evolving cryptosystems in the public key setting. Our focus has been on two approaches that can be applied to both public key encryption and digital signature schemes: forward security and intrusion resilience.

There are other key evolving cryptosystems in the public key setting that are of interest. For example, 'tamper evident' signature schemes (Itkis, 2003) are similar to forward secure signature schemes, but the secret signing key evolves in a truly unpredictable way (using a source of real randomness as one of the inputs). A public 'divergence test' indicates whether or not two signatures were created with signing keys from the same evolutionary path. As long as the true signer continues to sign messages honestly after the key exposure, the existence of a forging attacker can be detected (although it is not possible to tell which are the forgeries and which are the honest signatures).

Some researchers have considered key evolving cryptosystems where the public key is allowed to evolve (Naccache et al., 2001; Itkis and Xie, 2003). This can protect against key exposure that is coercive, for example, if an attacker demands your secret key at some point in time.

The recent survey by Itkis (2006) is recommended for other key evolving cryptographic constructs, applications and insights.

Acknowledgements

The author would like to thank NSF and the Packard Foundation for their support of his research; the students of ECS 228 at UC Davis for helping to debug the lecture notes from which this survey was drawn; Dimitri DeFigueiredo for drawing the figures; and Jonathan Katz and Moti Yung for many helpful comments.

References

Abdalla, M. and Reyzin, L. (2000) 'A new forward-secure digital signature scheme', *Advances in Cryptology – Asiacrypt 2000, LNCS 1976*, Springer, pp.116–129.

Anderson, R. (1997) Invited Lecture at Fourth ACM Confecence. Computer and Communication Security, 1997. Revised version in '*Two Remarks on Public Key Cryptology', Technical Report UCAM-CL-TR-549*, University of Cambridge, December 2002.

Back, A. (1998) 'Non-interactive forward secrecy', *Cypherpunks Mailing List 6/19/96.*

Bellare, M. and Miner, S. (1999) 'A forward-secure digital signature scheme', *Advances in Cryptology – Crypto '99, LNCS 1666*, Springer, pp.431–448.

Boneh, D., Boyen, X. and Goh, E. (2005) 'Hierarchical identity based encryption with constant size ciphertext', *Advances in Cryptology – Eurocrypt 2005, LNCS 3494*, Springer, pp.440–456.

Boneh, D. and Franklin, M. (2003) 'Identity-based encryption from the Weil Pairing', *SIAM Journal of Computing*, Vol. 32, No. 3, pp.586–615.

Canetti, R., Halevi, S. and Katz, J. (2003) 'A forward-secure public-key encryption scheme', *Advances in Cryptology – Eurocrypt 2003, LNCS 2656*, Springer, pp.255–271, Full version to appear in the *Journal of Cryptology.*

Cronin, E., Jamin, S., Malkin, T. and McDaniel, P. (2003) 'On the performance, feasibility, and use of forward-secure signatures', *ACM Conference Computer and Communication Security*, ACM Press, pp.131–144.

Desmedt, Y. and Frankel, Y. (1989) 'Threshold cryptosystems', *Advances in Cryptology – Crypto '89, LNCS 435*, Springer, pp.307–315.

Dodis, Y., Katz, J., Xu, S. and Yung, M. (2002) 'Key-insulated public key cryptosystems', *Advances in Cryptology – Eurocrypt 2002, LNCS 2332*, Springer, pp.65–82.

Dodis, Y., Katz, J., Xu, S. and Yung, M. (2003) 'Strong key-insulated signature schemes', *Public Key Cryptography – PKC 2003, LNCS 2567*, Springer, pp.130–144.

Dodis, Y., Franklin, M., Katz, J., Miyaji, A. and Yung, M. (2003) 'Intrusion-resilient public-key encryption', *Proceedings of RSA Conference – Cryptographer's Track, LNCS 2612*, Springer, pp.19–32.

Fujisaki, E. and Okamoto, T. (1999) 'Secure integration of asymmetric and symmetric encryption schemes', *Advances in Cryptology - Crypto '99, LNCS 1666*, Springer, pp.537–554.

Gentry, C. and Silverberg, A. (2002) 'Hierarchical ID-based cryptography', *Advances in Cryptology Asiacrypt 2002, LNCS 2501*, Springer, pp.548–566.

Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H. and Yung, M.(1997) 'Proactive public key and signature systems', *Fourth ACM Conference Computer and Communication Security*, ACM Press, pp.100–110.

Itkis, G. (2002) 'Intrusion-resilient signatures: generic constructions, or defeating strong adversary with minimal assumptions', *Conference Security in Communication Networks (SCN '02, LNCS 2576)*, Springer, pp.102–118.

Itkis, G. (2003) 'Cryptographic tamper evidence','*Tenth ACM Conference Computer and Communication Security*, ACM Press, pp.355–364.

Itkis, G. (2006) 'Forward security – adaptive cryptography: time evolution', *Handbook of Information Security*, Vol. 3, Chapter 199, H. Bidgoli (Ed). Wiley Publishers.

Itkis, G. and Reyzin, L. (2001) 'Forward-secure signatures with optimal signing and verifying', *Advances in Cryptology – Crypto 2001, LNCS 2139*, Springer, pp.332–354.

Itkis, G. and Reyzin, L. (2002) 'SiBIR: intrusion-resilient signatures, or towards obsoletion of certificate revocation', *Advances in Cryptology – Crypto '02, LNCS 2442*, Springer, pp.499–514.

Itkis, G. and Xie, P. (2003) 'Generalized key-evolving signature schemes or how to foil an armed adversary', *Conference. Applied Cryptography and Network Security, LNCS 2846*, Springer, pp.151–168.

Kozlov, A. and Reyzin, L. (2002) 'Forward-secure signatures with fast key update', *Security in Communication Networks, Third International Conference, LNCNS 2576*, Springer, pp.241–256.

Krawczyk, H. (2000) 'Simple forward-secure signatures from any signature scheme', *Proceedings of the Seventh ACM Conference Computer and Communication Security*, ACM Press, pp.108–115.

Malkin, T., Micciancio, D. and Miner, S. (2002) 'Efficient generic forward-secure signatures with an unbounded number of time periods', *Advances in Cryptology Eurocrypt '02, LNCS 2332*, Springer, pp.400–417.

Malkin, T., Obana, S. and Yung, M. (2004) 'The hierarchy of key evolving signatures and a characterization of proxy signatures', *Advances in Cryptology Eurocrypt 2004, LNCS 3027*, Springer, pp.306–322.

Naccache, D., Pointcheval, D. and Tymen, C. (2001) 'Monotone signatures', *Financial Cryptography, LNCS 2339*, Springer, pp.305–318.

Ostrovsky, R. and Yung, M. (1991) 'How to withstand mobile virus attacks', *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, ACM Press, pp.51–59.