Chapter 6

Authentication of Transit Flows

In this chapter, we exploit the unique features of the k-sibling Intractable hashing method in presenting two authentication schemes. In the first scheme, we propose a method which enables intermediate nodes in IP communication networks to verify the authenticity of transit flows. In the second scheme, we introduce a new one-time digital signatures scheme.

6.1 Introduction

There has been considerable interest in group-based applications over the last few years. There has also been a remarkable increase in real-time applications such as online video/audio streaming which have special quality of service requirements. As far as the security of these applications is concerned, new challenges (see Chapter 3) in designing security protocols for these applications have arisen. Usually, these applications have special quality-of-service (QoS) requirements, and the security services should be performed within its limits. One of the important security services is source authentication. Typical authentication schemes such as digital signatures have both high computational and space overhead, and hence they do not fulfill the new requirements of these applications. On the other hand, Message Authentication Codes (MAC) are more efficient, but they do not provide non-repudiation service. Therefore, new techniques are required which can not only guarantee secure communication, but also maintain the efficiency of the application. This problem has been well defined and explored in the literature and several techniques have been proposed [31, 64, 122, 121, 133, 164].

In this chapter, we continue the work in the direction of developing efficient cryptographic solutions for the problem of authentication in group communication. We introduce new authentication schemes that are based on the idea of the k-sibling intractable function family SIFF [170]. SIFF is a generalization of the universal one-way

function family theorem (see also [18]). It has the property that, given a set of initial strings colliding with one another, it is infeasible to find another string that would collide with the initial strings. This cryptographic concept has many useful applications in security and we have used it to develop new authentication scheme. In this chapter, we start by expanding the idea of SIFF to hierarchical SIFF. Then, we propose a scheme for authenticating 'transit' flows in IP communication. To our best knowledge, this topic has not been discussed elsewhere. Further, we propose a new one-time signature scheme that is efficient in generation and verification of signatures and with minimum space overhead which is suitable for end-to-end real-time applications.

This chapter is structured as follows. In the next section, we first illustrate the idea of k-SIFF and then expand it into a Hierarchical k-SIFF. In Section 6.3, a scheme for authenticating transit flow in communication networks is illustrated. In Section 6.4, the k-sibling one-time signature is presented.

6.2 K-Sibling Intractable Hashing

The construction and security properties of k-sibling intractable hash functions are discussed in [170]. Briefly, let U$= \cup_n U_n$ be a family of functions mapping $l(n)$ bit into $m(n)$ bit output strings. For two strings $x, y \in \sum^{l(n)}$ where $x \neq y$, we say that x and y collide with each other under $u \in U_n$, or x and y are siblings under $u \in \mathcal{U}_n$, if $u(x) = u(y)$.

In other words, sibling intractable hashing provides hashing that collides for k messages selected by the designer. It can be seen as the concatenation of two functions: universal hash function and collision-resistant hash function. More formally, we say that a family of universal hash functions

$$\mathcal{U} = \{U_n : n = \mathcal{N}\}$$

holds the k-collision accessibility property if, for a collection $\mathcal{X} = \{x_1, \ldots, x_k\}$ of k random input values $U_n(x_1) = \ldots = U_n(x_k)$ where $U_n : \{0,1\}^{\ell(n)} \rightarrow \{0,1\}^{L(n)}$ and $\ell(n), L(n)$ are two polynomials in n (n is the security parameter and $\mathcal{N}$ is the set of all natural numbers). A family of collision resistant hash functions

$$\mathcal{H} = \{H_n : n = \mathcal{N}\}$$

consists of functions that are one-way, and finding any pair of colliding messages is computationally intractable. k-sibling intractable hash functions can be constructed

as

$$kH_n = \{h \circ u : h \in H_n, u \in U_n\}$$

where $U_n : \{0,1\}^{\ell(n)} \to \{0,1\}^{L(n)}$ and $H_n : \{0,1\}^* \to \{0,1\}^{L(n)}$ (the notation $\{0,1\}^*$ stands for strings of arbitrary length). U_n is a collection of polynomials over $GF(2^{\ell(n)})$ of degree k.

6.2.1 Hashing with a Single Polynomial

The designer of a k-sibling intractable hash function first takes an instance of a collision intractable hash $H : \{0,1\}^* \to \{0,1\}^\ell$ (such as SHA-1) and a collection of k messages $\{m_1, \ldots, m_k\}$ that are to collide. Next, she computes

$$x_i = H(m_i)$$

randomly chooses $\alpha \in GF(2^\ell)$ and determines a polynomial $U : \{0,1\}^\ell \to \{0,1\}^\ell$ such that

$$U(x_i) = \alpha \text{ for } i = 1, 2, \ldots, k.$$

This can be done using the Lagrange interpolation. Having k points (x_i, α); $i = 1, \ldots, k$, it is easy to determine such a polynomial $U(x)$ which is different from a straight line. Note that $k+1$ points are needed to determine a polynomial of degree k.

Denote $\mathbf{H} = U \circ H$. By construction $\mathbf{H}(m_i) = \alpha$ for all $i = 1, 2, \ldots, k$.

The hash function $\mathbf{H}$ can be characterized by the following properties:

- finding collisions (those incorporated by the designer in U as well as those existing in H) is computationally intractable, assuming the attacker has the descriptions of the two functions H and U. The descriptions must be available in a public, read-only registry. Note that the description of U takes $k+1$ values from $GF(2^\ell)$.

- the hash function treats messages as an unordered collection of elements. To introduce an order, the designer needs to calculate $\mathbf{H}$ for an ordered sequence of messages so that any message $m_i = (i, m_i')$ where i indicates the position of the message in the sequence. In other words, $\mathbf{H}(i, m_i') = \alpha$ for all $i = 1, \ldots, k$.

Note that, if the number of colliding messages is large (say $k > 1000$), then to compute hash values, one would need to fetch $k + 1$ coefficients of polynomials $U(x)$. This introduces delays. Is there any other way to design k-sibling hashing?

6.2.2 Hierarchical Sibling Intractable Hashing

Given k-sibling intractable hash function $\mathbf{H}^{(k)}$ and a set $\mathcal{M} = (m_1, \ldots, m_{k^2})$ of messages. A k^2-sibling intractable hash function denoted as

$$\mathbf{H}^{(k^2)} = \mathbf{H}^{(k)} \circ \mathbf{H}^{(k)}$$

is a collection of $k+1$ k-sibling intractable hash functions where

$$\mathbf{H}_i = U_i \circ H \text{ with collisions in } \mathcal{M}_i = (m_{ik+1}, \ldots, m_{(i+1)k})$$

for $i = 0, \ldots, k-1$, and

$$\mathbf{H}_k = U_k \circ H \text{ with collisions in } \mathcal{X} = \{h_i = \mathbf{H}_i(\mathcal{M}_i); i = 1, \ldots, k\}.$$

To find the hash value of a message, it is not necessary to know all polynomials U_i. For a message $m \in \mathcal{M}_i$, it is sufficient to know two polynomials only, namely, U_i and U_k.

In general, sibling intractable hashing with k^r colliding messages can be defined as

$$\mathbf{H}^{(k^r)} = \mathbf{H}^{(k)} \circ \mathbf{H}^{(k^{r-1})}$$

for $r > 2$. Similarly, to compute a hash value for a single message, it is necessary to learn r polynomials of degree k.

The polynomials $U_{i,j}(x)$ are in fact arranged in a tree structure. The leaves of the tree are $U_{1,j}$ for $j = 1, \ldots, k^{r-1}$. The next layer is created by polynomials $U_{2,j}$; $j = 1, \ldots, k^{r-2}$ and so on. The root is $U_{r,1}$. Figure 6.1 illustrates the concept graphically.

6.3 Authentication of Packets

Message authentication is an important service in information security. Typical authentication schemes such as digital signatures use public-keys, while Message Authentication Codes (MAC) use private keys. Digital signatures are known for their high computation overhead, while MAC does not provide non-repudiation service. In cases such as in IP communication, we may have a stream of independent messages to be authenticated. Neither the typical digital signatures provides efficient solution, nor MAC provides enough security service. Therefore, new techniques are required to provide both security and efficiency.

A further motivation is the requirement by the intermediate nodes in IP network for a technique to authenticate the packets in their transitions from source to destination. IPSec [90] is a security mechanism designed to provide security services for IP

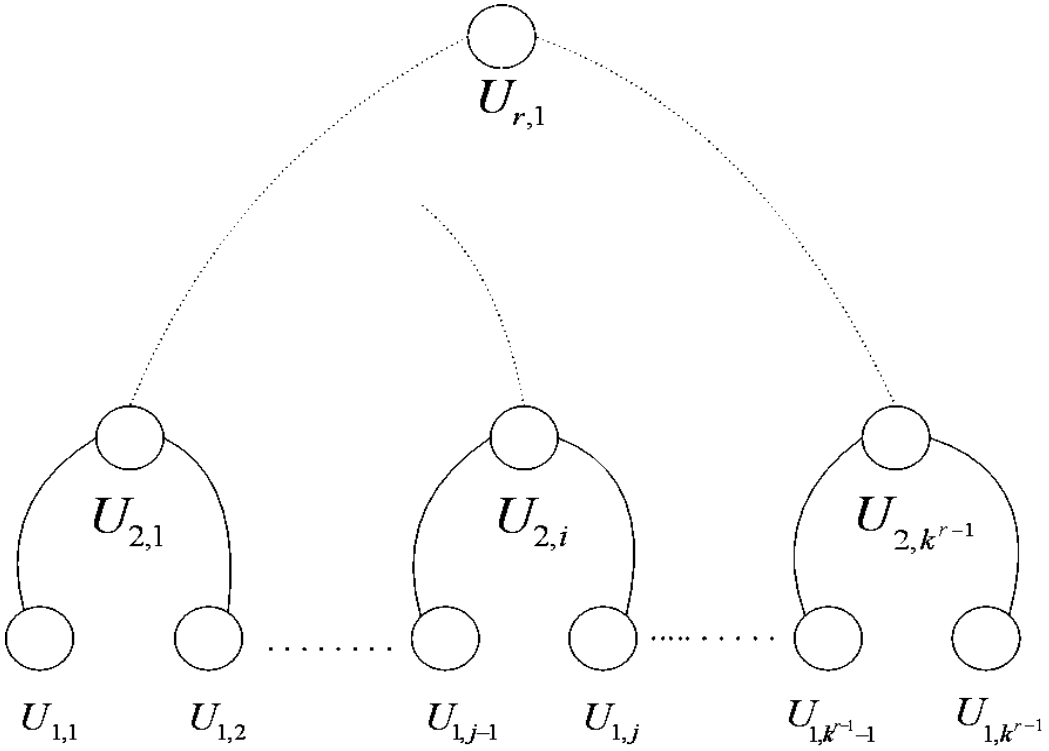

Figure 6.1: Hierarchical Sibling Hashing

communication. It provides source authentication, confidentiality, as well as integrity. As far as source authentication is concerned, with the symmetric authentication option provided by IPSec, only hop-by-hop authentication can be achieved. This means that a node that receives a message only knows that it has originated from an authenticated node when they share a common key in the domain. However, it would not be possible for intermediate nodes along the path to check the authenticity of the messages. In doing this, it would be possible to discover harmful actions such as a denial-of-service attack in their early stages, before they are propagated to the destination. We seek a mechanism that enables intermediate nodes to verify the source of the message.

Possible solutions are: for each message to be given a tag independent of the others, or for the concatenation of all messages to be given a single common tag. In the first method, the resulting tags may prove too impractical to be maintained, while in the second method the validation of one message requires the use of all other unrelated messages in recalculating the tag. A preferable method would be one that employs a single common tag for all the messages in such a way that a message can be verified individually without involving other messages. This can be achieved by using SIFF, in which all messages are represented as a string of $l(n)$ bits long.

6.3.1 Authentication with Single Hashing

The security goal is to enable interested parties of the network (nodes) to authenticate messages (packets) in transit. A natural restriction imposed on authentication is that packets of the same message (generated by the same source) may travel through different routes. In effect, a node may see a small subset of all packets generated by the source. Those that are seen do not typically follow the initial order. Note that authentication of packets based on some sort of chaining is useless. Our solution is based on sibling intractable hashing.

Model

The scheme consists of the following cryptographic primitives:

- a secure signature scheme $SG = (S_{sk}, V_{pk}, G())$ for message authentication S_{sk} is the signing algorithm that for a given message m and a secret key sk produces a signature or $s = S_{sk}(m)$, V_{pk} is the verification algorithm that for a public key pk and a signature s returns 1 if the signature is valid and 0 if otherwise. $G()$ generates a pair of keys: sk, pk. The meaning of "secure" will not be discussed here; the interested reader can consult relevant papers (see [14])
- a collision intractable hash function H, $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$
- n-sibling intractable hash function $\mathbf{H}^{(n)}$
- a Public Key Infrastructure (PKI) that provides on demand authenticated public keys of all potential senders (normally in the form of certificates)

Sender

1. The source (sender) takes a message M and splits it into n packets (datagrams). In other words, $M = (m_1, \ldots, m_n)$ where m_i is the i-th datagram,
2. Computes $x_i = H(m_i)$ for all $i = 1, \ldots, n$,
3. Chooses randomly α,
4. Computes the coefficients u_i from the polynomials $U(x_i) = \alpha$ using Lagrange interpolation for all $i = 1, \ldots, n$,
5. Designs an SIFF instance for the message M such that:

$$H^{(n)}(m_1) = H^{(n)}(m_2) = \ldots = H^{(n)}(m_n)$$

6. Takes the sibling intractable hash $\mathbf{H}^{(n)}$ and computes the signature of the message M as

$$s = S_{sk}(H(H^{(n)}(M) \| H(u_0, \ldots, u_n)), \text{timestamp})$$

where

$$U(x) = u_0 + u_1 x + \ldots + u_n x^n; u_i \in GF(2^\ell),$$

and

$$H^{(n)}(M) = \alpha,$$

7. Puts the signature together with coefficients of $U(h)$ into a read-only registry R accessible to everybody.

Verifier

1. Receives datagrams m_i and computes $h_i = H(m_i)$ where $i \in \{1, \ldots, n\}$,
2. Contacts the registry R and fetches the signature, coefficients u_i, and α
3. Obtains the authenticated public key pk of the sender from the PKI facility,
4. Checks the validity of the signature using the algorithm $V_{pk}(s)$, if it is Ok, continue, otherwise, abort $\perp$,
5. Recovers the polynomial $U(x)$ from the coefficients u_i and α and form the polynomial as in step 6 of the signing
6. Computes $h' = U(h_i)$,
7. Checks validity of $h_i = h'$, accept if it is OK; otherwise $\perp$.

Security

The security of the scheme is basically evaluated from the security of underlining components: digital signature, hash function, and SIFF algorithms. However, one has to note that the polynomial $U(x)$ must be used by the sender to produce the final hash value that is signed by the sender. This is done to prevent manipulation with the structure of the sibling intractable hash function $\mathbf{H}^{(n)}$. The signature is used to prove the genuineness of the polynomial components that are going to be fetched from the registry by the verifier.

Efficiency

Note that the verification of the first datagram is the most expensive as it will take verification of signature (one exponentiation if signature is based on RSA or ElGamal) that also involves calculation of hash $h_i = H(m_i)$, computation of $U(h_i)$, and evaluation of $H(u_0, \ldots, u_n)$. Any new message can be verified using one extra evaluation of H and U. It computes the hash $h_i^{'} = H(m_i)$ and computes $h^{'} = U(h_i^{'})$. If $h^{'} = h$ then, it accepts the message; otherwise, it rejects it. As far as communication is concerned, the verifier must fetch the signature and the polynomial $U(x)$. Note that the length of $U(x)$ is almost the same as the whole message M. This seems to be the weakest point of the construction.

6.3.2 Authentication with Hierarchical Hashing

In this case, the sibling intractable hashing is computed using a family of polynomials $U_{i,j}$ with $i = 1, \ldots, r$ and $j = 1, \ldots, k^{r-i}$. The message consists of $n = k^r$ datagrams. To compute $H^{(n)}(M)$ it is enough to fetch r polynomials of degree k (that is in a sense, a path between a leaf and the root). If we choose k=2, then the verifier needs to fetch $3 \times \log_2 n$ coefficients. With pre-determined single points for the polynomials, the number can be reduced to $2 \times \log_2 n$ without security deterioration. Figure 6.2 illustrates the idea graphically.

The tree of polynomials must also be subject to hashing (to make the verifier sure that she uses the correct instance of the sibling intractable hash). One good feature is that the verifier would like to use explicitly all polynomials she has imported from R. The signer may help the verifier by first using parallel hashing [43] (i.e. using more than one processor to compute the hash value concurrently) for the polynomials and storing in R all intermediate results of hashing. The verifier puts the polynomials together with intermediate hash values to generate $H(U)$ where U means collection of all polynomials.

The advantage of hierarchical hashing is evident when we consider the storage required to allow authentication of k public values using the following approach. An entity A authenticates t public values $Y_1, Y_2, \ldots, Y_t$ by registering each with a read-only registry or trusted third party. This approach requires registration of t public values, which may raise storage issues at the registry when t is large. In contrast, a hierarchical hashing requires only a single value registered in the registry. If a public

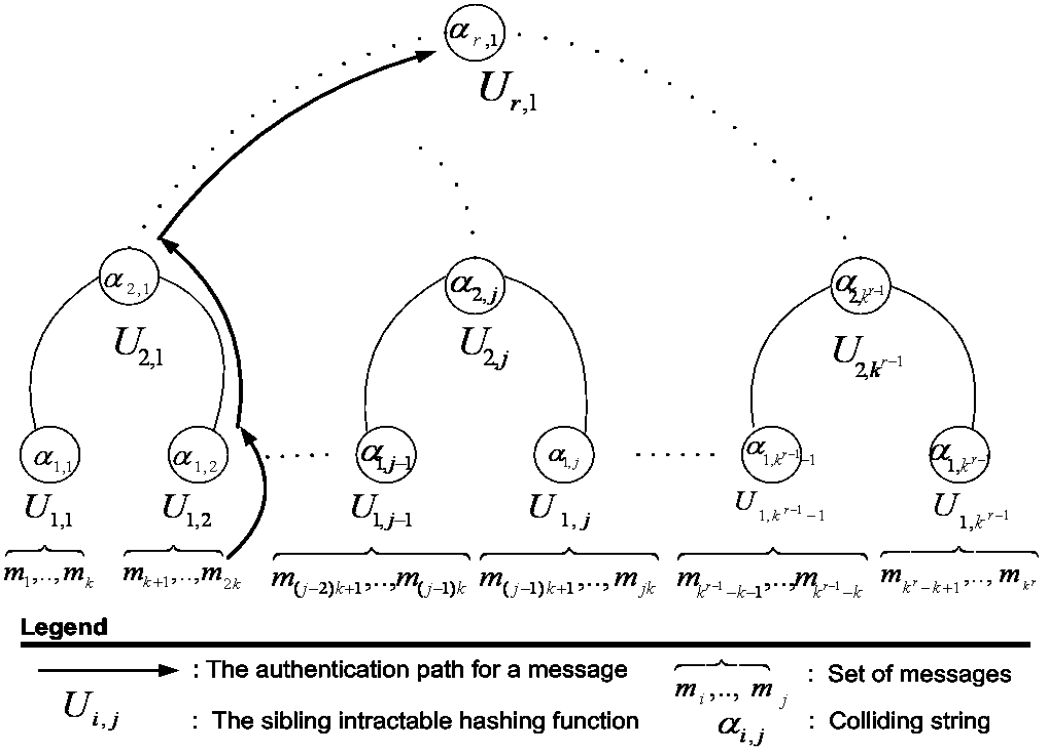

Figure 6.2: Authenticating messages with Hierarchical Sibling Hashing

key Y_i of an entity A is the value corresponding to a leaf in an authentication tree, and A wishes to provide B with information allowing B to verify the authenticity of Y_i, then A must (store and) provide to B both Y_i and all hash values associated with the authentication path from Y_i to the root. In addition, B must have prior knowledge and trust in the authenticity of the root value R. These values collectively guarantee authenticity, analogous to the signature on the public-key certificate. The number of values each party must store is $log(t)$.

Consider the length (or the number of edges in) the path from each leaf to the root in a binary tree. The length of the longest such path is minimized when the tree is balanced, that is, when the tree is constructed such that all such paths differ in length by at most one. The length of the path from leaf to the root in a balanced binary tree containing t leaves is about $log(t)$.

Using a balanced binary tree as authentication tree, with t public values as leaves, and authenticating tree with t public values as leaves, authenticating a public value may be achieved by hashing $log(t)$ values along the path to the root.

Authentication trees require only a single value which is the root value, in a tree to be registered as authentic, but verification of the authenticity of any particular leaf

value requires access to and hashing all values along the authentication path from leaf to root.

To change a public (leaf) value or add more values to an authentication tree requires re-computation of the label on the root vertex. For a large balanced tree, this may involve a substantial computation. In all cases, re-establishing trust of all users in this new root value is necessary.

The computational cost involved in adding more values to a tree may motivate constructing the new tree as an unbalanced tree with the new leaf value being the right child of the root, and the old tree being the left. Another motivation for allowing unbalanced trees arises when some leaf values are referred far more frequently than others.

6.3.3 Security Issues

There follow some remarks on the security of the schemes:

- the scheme signs simultaneously all datagrams using a single signature. The important difference of this scheme from other schemes is that verification of datagrams can be done independently (or in parallel). In other words, to authenticate datagrams, one does not need to know all datagrams,
- no authentication is required for the coefficients fetched from a read-only registry (assuming coefficients have been fetched from the 'right' registry; otherwise, an enemy can fake the packets). This is because, if entries are tampered with, then packets will be rejected since the final hash recovered from the signature will be different from the hash value obtained from the datagrams and the polynomial,
- the only security problem could be of denial-of-service attack when an enemy may intentionally modify polynomial coefficients to reject the datagrams,
- in both flat and hierarchical k-sibling approaches, a single signature is required:
  1. the description of public polynomial coefficients used in the k-sibling intractable hashing takes about n integers each of size 160 bits for SHA-1, where n is the number of packets of the message M and $k > 2$. If k=2, then this number = $2n$.
  2. the scheme may be used against denial-of-service attacks. In particular, it would be possible for receivers at the intermediate nodes to ignore those packets that have failed to pass k-sibling hashing verification.

- the authentication scheme described above could be used for both types of IP data transfer modes: connection-oriented and connectionless. In the case of connection-oriented communication, where a node or destination sees almost all the packets of the message, flat sibling hash with a single polynomial $U(x)$ of degree n is best applicable. If, however, a node may see only a small fraction of packets, as in connectionless communication, then the hierarchical sibling with 2-sibling hashing seems to be superior.

6.4 K-Sibling One-time Signature

One-time signatures derived their importance from their fast signature verification, in contrast to typical digital signature schemes, which have either high generation or verification computation time. One-time signatures are a perfect option for authenticating particular types of applications where receivers are short of computing resources, such as chipcards, or for online applications, such as video/audio streaming, which requires fast verification, as well as for centralized applications such as voting systems.

Lamport [97], Rabin [128], Merkle [108] and GMR [66] are well known examples of one-time signature schemes. Although they differ in their approaches, they share the same idea: only one message can be signed using the same key. Once the signature is released, its private key is not used again; otherwise, it would be possible for an adversary to compute the key, and hence given its name one-time signature. A more description is found in Chapter 7.

One-time signatures have to be efficient and secure. Typically, the verification of the signature is expected to be very efficient. Additionally, signatures have to be initialized well ahead of the time when messages are to be signed and verified. This allows the signer to pre-compute the signature parameters so that they can be fetched by potential verifiers. Once the message is known, the signer can sign it quickly, and receivers can verify the signed message efficiently. A distinct characteristic of one-time signatures is that they are used once only. To sign a new message, the signer must initialize the signature parameters (parameters of old signatures must not be reused). The security of one-time signatures is measured by the difficulty of forging the signature by an adversary who normally has access to a single pair: a message and its signature.

The main advantage of one-time signatures is that they only rely on one-way functions without trapdoors. This can be implemented using a fast hash function such

as SHA-1. One of the new approaches in designing such signatures is the BiBa one-time signature [121]. BiBa is an acronym for BIns and BAlls. It uses the bins and balls analogy to create a signature. To sign a message m, the signer first uses random precomputed values generated in a way that a receiver can authenticate them with a public key. These precomputed values are called SEALS (SElf Authenticating vaLueS). The signer then compute the hash of the message $h = H(m)$), and then computes the hash function G_h. Now, the collision of SEALS under a hash function G_h forms a signature: $G_h(s_i) = G_h(s_j)$, where $s_i \neq s_j$. The BiBa signature exploits the birthday paradox property, in that the signer who has a large number of balls finds a collision (signature) with high probability, but a forger who only has a small number of balls has a negligible probability of finding a signature.

The BiBa signature scheme has desirable features such as small authentication space overhead and fast verification time. However, its public keys are very large, the time needed to generate a signature is higher than any other known system, and it requires parallel processors to find collision of SEALS. This makes signature generation a large computation overhead. Also, it uses an *ad hoc* approach to find collisions among the 'SEALS' to the corresponding bin, which results in high signature generation time.

6.4.1 The Scheme

We propose a variant approach to BiBa by using the SIFF method. SIFF provides hashing with a controlled number of easy-to-find collisions. In other words, we apply a *deterministic* approach in finding a collision (signature). As for signatures based on public-key cryptography, we assume that we are going to produce signatures for digests of messages. Thus, suppose that messages to be signed are of constant length (160 bits if we use SHA-1).

Let $\mathrm{SIFF}_i(x)$ be an instance of k-sibling hash function that for k inputs $x_{i,0}, \ldots, x_{i,k-1}$ produces the output α_i or

$$\mathrm{SIFF}_i(x_{i,j}) = \alpha_i \text{ for } j = 0, \ldots, k-1$$

The function applies a polynomial $U_i(x) = u_{i,0} + u_{i,1}x + \ldots + u_{i,k-1}x^{k-1}$ that collides for the inputs $x_{i,0}, \ldots, x_{i,k-1}$ or

$$U_i(H(x_j)) = \alpha_i$$

where H is a collision-resistant hash function. Assume that the message to be signed is $M = (m_1, \ldots, m_t)$ where m_i are v-bit sequences. The message M consists of vt

bits (typically of the length 160 bits). To design our one-time signature we use the sequence of t instances of SIFF where each instance applies 2^v collisions, as shown in the following algorithm:

Initialization

The signer builds up the sequence of $\mathrm{SIFF}_i(x)$ for $i = 1, \ldots, t$. He starts from $\mathrm{SIFF}_1(x)$. First he picks up at random a sequence of 2^v integers (whose length is determined by the security parameter of the signature). Let the sequence be $r_{1,j}$; $j = 0, \ldots, 2^v - 1$ and denote $x_{1,j} = (r_{1,j}, j)$. The signer chooses at random the output α_1 and calculates the polynomial $U_1(x)$ such that

$$U_1(H(x_{1,j})) = \alpha_1 \text{ for } j = 1, \ldots, 2^v - 1$$

Next, he creates $\mathrm{SIFF}_i(x)$ for $i = 2, \ldots, t$. For each i, he selects at random integers $(r_{i,j})$; $j = 0, \ldots, 2^v - 1$, composes

$$x_{i,j} = (r_{i,j}, j, \alpha_{i-1})$$

and calculates the polynomial $U_i(x)$ such that

$$U_i(H(x_{i,j})) = \alpha_i \quad \text{for } j = 1, \ldots, 2^v - 1$$

for a random α_i. The polynomials $U_i(x)$ and the final value α_t are made public in the read-only authenticated registry; $i = 1, \ldots, t$.

Signing

Given a message $M = (m_1, \ldots, m_t)$. The signer marks the input x_{1,m_1} and extracts r_{1,m_1} and similarly determines r_{i,m_i} for $i = 2, \ldots, t$. The signature is

$$S(M) = (r_{1,m_1}, \ldots, r_{t,m_t})$$

The pair $(M, S(M))$ is the signed message.

Verification

The verifier takes the pair $(\tilde{M}, S(\tilde{M}))$ and the public information, i.e. coefficients of polynomials $U_i(x)$ and α_t. Knowing $\tilde{x}_{1,\tilde{m}_1} = (\tilde{r}_{1,\tilde{m}_1}, \tilde{m}_1)$ and the polynomial $U_1(x)$, he can compute $\tilde{\alpha}_1$. Next, he recreates the inputs $\tilde{x}_{i,\tilde{m}_i} = (\tilde{r}_{i,\tilde{m}_i}, \tilde{m}_i, \tilde{\alpha}_{i-1})$ for $i = 2, \ldots, t$. If the last recovered $\tilde{\alpha}_t$ is equal to α_t recovered from the registry, the signature is considered valid. Otherwise, it is rejected.

6.4.2 Security Issues

Suppose that an adversary knows a signed message and tries to modify either message or signature such that the forged (and signed) message passes verification. Obviously, the adversary also knows the public information. Informally, if the adversary is successful it means he was able to create either a new collision (which was not designed by the signer) or was able to guess one of the strings $r_{i,m'}$. The first event is excluded if we assume that the SIFF is collision resistant. The probability of the second event can be made as small as required by choosing an appropriate length of the strings $r_{i,j}$. It is important to note that the above considerations are valid only if the public information about signatures is authentic.

Definition 6.1 Existential Forgery on k-sibling one-time signature

Let v be the security parameter of the signature. We say that the probability of an existential forgery (in polynomial time) of a k-sibling one-time signature is $p(v)$ if there exists a probabilistic adversary algorithm $\mathcal{A}$ having as the input:

- *the function H*
- *the polynomials $U_1, U_2, ..., U_t$*
- *the value α_t*
- *the observed message $M = (m_1, m_2, ..., m_t)$ and its valid signature $S(M) = (r_{1,m_1}, r_{2,m_2}, ...r_{t,m_t})$*

which can produce, in polynomial time in v and with probability $p(v)$, a message $M' = (m'_1, m'_2, ..., m'_t) \neq M$ and its valid signature $S(M) = (s_1, s_2, ..., s_t)$

Theorem 6.1 *For each polynomial Q, there is a value v of the security parameter, such that the probability of an existential forgery (in polynomial time) of a k-sibling one-time signature is less than $1/Q(v)$.*

Proof. Let $Q(v)$ be a polynomial where v is the security parameter of the signature. Since all $U_i \circ H, i = 1 \ldots t$; are $SIFF$, and therefore one-way functions, there exists a value v such that $Pr(U_i^v \circ H(x) = U_i^v \circ H(\mathcal{B}(U_i \circ H(x))) < 1/Q(v)$ for each i, each x and for each probabilistic polynomial time algorithm $\mathcal{B}$.

Assume a probabilistic adversary algorithm $\mathcal{A}$ as described in the existential forgery definition.

If we denote:

$$\begin{aligned}\beta_1 &= U_1(H((s_1, m'_1)))\\ \beta_2 &= U_2(H((s_2, m'_2, \beta_1)))\\ \beta_3 &= U_3(H((s_3, m'_3, \beta_2)))\\ &\vdots\\ \beta_t &= U_t(H((s_t, m'_t, \beta_{t-1})))\end{aligned}$$

then $\beta_t = \alpha_t$.

Denote $x_1 = (r_{1,m_1}, m_1)$, $y_1 = (s_1, m_1)$, and for $2 \leq w \leq t$, $x_w = (r_{w,m_w}, m_w, \alpha_{w-1})$ and $y_w = (s_w, m_w, \beta_{w-1})$. We consider two cases:

Case 1: There is an index $w \geq 1$ such that $\beta_w = \alpha_w$ and $\beta_{w-1} \neq \alpha_{w-1}$. Then $y_w \neq x_w$.

Case 2: For all $i = 1, 2, ..., t$, $\beta_i = \alpha_i$. Since $M \neq m$, there is an index $1 \leq w \leq t$ such that $y_w \neq x_w$.

In both cases, there is an index w such that $y_w \neq x_w$.

We design a probabilistic adversary algorithm $\mathcal{B}$ which proceeds as follows:

1. simulates $\mathcal{A}$ and produces v polynomial $p(v)$ a message $m = (m'_1, m'_2, ..., m'_t)$ and its valid signature $S(m') = (s'_1, s'_2, ..., s'_t)$

2. continues by computing

$$\begin{aligned}\alpha_1 &= U_1(H((r_{1,m_1}, m_1)))\\ \beta_1 &= U_1(H((s_1, m'_1)))\\ \alpha_2 &= U_2(H((r_{2,m_2}, m_2, \alpha_1)))\\ \beta_2 &= U_2(H((s_2, m'_2, \beta_1)))\\ &\vdots\\ \alpha_{w-1} &= U_w(H((r_{w,m_w}, m_{w-1}, \alpha_{w-2})))\\ \beta_{w-1} &= U_w(H((s_{w-1}, m'_{w-1}, \beta_{w-2})))\end{aligned}$$

then

$$(r_{w,m_w}, m_w, \alpha_{w-1}) = x_w \neq y_w = (s_w, m'_w, \beta_{w-1})$$

and

$$U_W(H(y_w)) = U_W(H(x_w))$$

Step 2 is deterministic and all values can be computed in polynomial time. Hence, $\mathcal{B}$ can in polynomial time with probability $p(v)$ compute two values $x_w \neq y_w$ such that

$$U_w \circ H(x_w) = U_w \circ H(y_w)$$

The one-wayness of $U_w \circ H$ implies that there is a value of v such that $p(v) < 1/Q(v)$. □

6.4.3 Performance Issues

The signature allows trading-off efficiency of verification against the workload necessary to set up the signature system. This is a very important aspect of the signature. Note, however, that the setup is done for each single message (this is one-time signature). Verification is done many times, typically as many times as there are different recipients. Consider two extreme cases: the first with the longest signature, where SIFFs are designed for binary messages or $v = 1$ (t is the largest), and the second with $t = 1$. The first case permits a very efficient setup of the system with relatively small public information. The price to pay is the bandwidth necessary to transport a very long signature and verification consumes a large number of hash operations (as many as bits in the signed message).

The second case applies to a relatively small number of SIFFs (t is small). The setup is very expensive since any single SIFF applies large number of collisions, and in effect the corresponding polynomials are very long. Receivers must fetch polynomial coefficients for verification. Verification seems to be fast as it requires a small number of hash operations. Signatures are relatively short.

Some scope for more efficient implementation exists if the strings $r_{i,j}$ are generated differently. Note that, in fact the system applies $t2^v$ such strings, but only t are used as the signature. To reduce the necessary storage for keeping the strings by the signer, it is reasonable to choose at random $t + 1$ integers $r_{i,1}$; $i = 1, \ldots, t$ and the integer R. A polynomial $G(x)$ of degree t can be designed such that $G(0) = R$ and $G(i) = r_{i,1}$ for $i = 1, \ldots, t$. Note that other $r_{i,j}$ can be derived from the polynomial $G(x)$. This way of generation of $r_{i,j}$ is secure in the sense that signatures reveal t points on $G(x)$ leaving a single point on $G(x)$ unknown to the adversary.

6.5 Summary

The sibling intractable hashing function family, or SIFF is a useful crytographic primitive that might be used to solve a number of problems. In this chapter, we began by expanding the idea of SIFF from a single polynomial structure to a hierarchical structure. The later SIFF structure is more efficient than the former in terms of computing the hash values since it takes a tree shape, while the former takes a flat shape. We have exploited the hierarchical SIFF to design a new authentication scheme that can verify the authenticity of independent messages traversing from node to node in IP communication. In particular, we use the scheme to authenticate the source of packets at intermediate nodes in a communication network. This problem has not been discussed elsewhere in the literature. Also, we have used the SIFF to design a new one-time digital signature which has low computation and space overhead. Recently, one-time signatures have been the subject of research for authenticating group communication applications as an efficient authentication tool, and this will be considered in the next chapter.

Chapter 7

One-time Signatures for Authenticating Group Communication

One-time signatures are an important and efficient authentication utility. Various schemes already exist for classical one-way public-key cryptography. One-time signatures have not been sufficiently explored in the literature in the branch of society-oriented cryptography. Their particular properties make them suitable, as a potential cryptographic primitive, for broadcast communication and group-based applications. In this chapter, we contribute by filling this gap by introducing several group-based one-time signature schemes of various versions: with proxy, with trusted party, and without trusted party.

7.1 Introduction

As discussed earlier, one-time signatures are an important public-key cryptography primitive. They derive their importance from their fast signature verification. This is in contrast to the conventional digital signature schemes, which usually have high generation or verification computation time. One-time signatures can be an ideal option for authenticating particular types of applications where receivers are of low power capability, such as smart carts, or for online applications, such as video/audio streaming, which require fast verification, as well as for centralized-processing applications such as polling systems.

On the other hand, groups play an important role in contemporary communication. Numerous examples of group applications include the stock exchange, collaborative tasks, and many other multicast applications. Group communication has a (potentially) high communication overhead, and it is desirable that group members can authenticate their communications efficiently. Cryptographic transformations by a group of participants was the subject of investigation in so-called *society-oriented cryptography* ([52], [27]). Unlike classical cryptography, society-oriented cryptography

allows groups of cooperating participants to carry out cryptographic transformations. That is, society-oriented cryptography requires distribution of the power of performing a cryptographic transformation among a group of participants such that only designated subsets of the group can perform the required cryptographic operation, but unauthorized subsets cannot do so.

With the recent interest in securing group and broadcast communication, there has been a great demand for designing a new class of fast signature schemes that can handle a vast number of signatures from broadcast or group-based applications efficiently, rather than using typical signature schemes. Hence, there have been a number of attempts in society-oriented cryptography to design signature schemes for authenticating group-based scenarios.

Several schemes have been proposed that use classical authentication schemes such as digital signatures (RSA [132], ElGamal [57]) for group-based transformations. However, these conventional methods typically have a high computational overhead, and hence they may not fulfill the new requirements with regard to the efficiency of the emerging applications. Besides, the nature of authenticating online real-time applications usually requires fast authentication. That is, the extra potential complexity which is involved in the typical digital signatures to provide extra security adds more computational time. In contrast, one-time signatures provide the required security services with less computational overhead.

As mentioned, one-time signatures are potentially far more (computationally) efficient than classical authentication methods. This leads us to explore new ways of improving the efficiency of authentication in group communication using one-time signatures. That is, we attempt to apply one-time signatures for situations where the right to execute signature operations is shared by a group of signers. We propose a scheme for a threshold proxy signature. We achieve this by introducing a few intermediate schemes. We combine the concepts of one-time signature and threshold group signature by using the idea of proxy signing. Proxy one-time signature was first used in [91] to authenticate mobile agents in low-bandwidth communications. On the other hand, and to the best of our knowledge, the problem of finding a group oriented one-time signature has not been discussed elsewhere.

In this chapter, we begin by reviewing the relevant work in the area of one-time signatures. To allow application of several one-time signature schemes in a common way, we establish a construction model for the proposed protocols. To reach our goal, we investigate the problem of one-time signature in two scenarios: with a proxy signer

and with a group of signers. We start with one-time signature for the case when a signer wants to delegate his one-time signature to a proxy who would sign on his behalf. Then in Section 7.5, we show how it is possible to construct a threshold one-time signature using the Shamir secret sharing method. This may happen with or without the aid of a trusted party. Finally, in Section 7.6, we design a scheme for threshold proxy signature.

7.2 Related work

Lamport [97], Rabin [107], Merkle [108] and GMR [66] are well known examples of one-time signature schemes. They share the same basic idea, and are based on committing to secret keys via one-way functions. Rabin uses an interactive approach for verification of signatures with the signer. These schemes differ in their approaches, but they share the same idea: only one message can be signed using the same key. Once the signature is released, its private key is not used again; otherwise, it would be possible for an adversary to compute the secret key.

A new approach to designing such signatures is the BiBa one-time signature [121]. The BiBa signature exploits the birthday paradox property. A large number of secret keys is used to find collisions among the generated keys associated with the message. This way of signing requires a long pre-computational time. Reyzin and Reyzin [129] solve BiBa's disadvantage of having a very long signing time. Their idea is to calculate the number of required keys according to the size of the message and pre-determined length of the signature. Based on this, key generation would be very fast, and hence signing is faster.

One-time signatures have been used in group communication for authenticating streaming applications in multicast communication. Gennaro and Rohatchi [64] used a chained method with one-time signature. Rohatchi used a k-times one-time signature based on an on-line/off-line approach. Perrig used it in Tesla [122]. Al-Ibrahim *et al* in [7] introduced k-sibling one-time signature for authenticating transit flows. Wang *et al* in [162] used oblivious transfer protocol for designing proxy one-time signature.

7.3 A class of one-time signature schemes

In this section, we establish a model for one-time signature schemes. The model is not aimed at introducing a new kind of signature. We want to set a common view of several well-established signature schemes in order to be able to apply any one of them in our

subsequent scenarios. Therefore, not every signature scheme in our model is secure, and the properties of each such particular scheme are to be investigated separately.

Our model consists of a signer S and a verifier V and is described for a security parameter $K > 0$ as a tuple $\mathcal{O} = (M, X, Y, h, v, \pi)$ where M is the set of messages, X, Y are finite sets, $|X| > 2^K$, $h : X \to Y$ is a one-way hash function (we assume that for any polynomial Q the probability to find for a given $y \in Y$ in time $Q(K)$ a value $x \in X$ such that $y = h(x)$ is less than $Q(K)/2^K$) , $v \geq 1$ is an integer and $\pi : M \to 2^{\{1,2,\ldots,v\}}$ is a one-way function, where $M \to 2^{\{1,2,\ldots,v\}}$ is the set of values of M that are subsets of $1, 2, \ldots, v$. All parts of $\mathcal{O}$ are public. If a signer S sends a message $m \in M$ to a verifier V, the signature creation and verification proceeds as follows:

Key generation

Signer S

1. chooses v random values $s_1, s_2, \ldots, s_v \in X$ (the secret keys of the signer)
2. computes v values $p_1 = h(s_1)$, $p_2 = h(s_2), \ldots, p_v = h(s_v) \in Y$ and makes them public.

Signing

Signer S

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m), \quad (1 \leq r \leq v)$
2. sends m and the signature $(s_{j_1}, s_{j_2}, \ldots, s_{j_r})$ to the verifier V.

Verification

Verifier V

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. computes $h_1 = h(s_{j_1}), h_2 = h(s_{j_2}), \ldots, h_r = h(s_{j_r})$
3. accepts the signature if and only if $h_1 = p_{j_1}, h_2 = p_{j_2}, \ldots, h_r = p_{j_r}$.

The model includes schemes like Lamport [97] or Merkle [108] as special cases. The schemes of Rabin [128], GMR [66], BiBa [121] are of a different type. The "better than BiBa" scheme of Reyzin and Reyzin [129], Bos and Chaum [26], and HORS++ of [125] belong to this model.

7.4 A simple one-time proxy signature scheme

Delegation of rights is a common practice in the real world. A manager of an institution may delegate to one of his deputies the capability to sign on behalf of the institution while he is on holiday. For electronic transactions, a similar approach is needed to delegate the manager's digital signature to the deputy.

Proxy signature is a signature scheme where an original signer delegates his/her signing capability to a proxy signer, and then the proxy signer creates a signature on behalf of the original signer. When a receiver verifies a proxy signature, he verifies both the signature itself and the original signer's delegation. Mambo, Usuda and Okamoto (MUO) in [105] established models for proxy signatures. They classified proxy signatures, based on delegation type, as *full delegation*, *partial delegation*, and *delegation by warrant*. In full delegation, the signer gives his secret key to the proxy. In partial delegation, the signer creates a separate secret key for the proxy, but it is derived from his secret key. In signing with warrant, the signer signs the public key. In addition, they provide various constructions for proxy signature schemes with detailed security description and analysis. Their proxy signatures provide various security services including:

- **Unforgeability**. Only the proxy signer (besides the original signer) can create a valid signature for the original signer.
- **Proxy signer's deviation**. Each valid proxy signer's signature can be detected as her signature.
- **Verifiability**. A positive verification of a proxy's signature guarantees that the original signer has delegated the power of signing to the proxy.
- **Distinguishability**. A valid proxy's signature can be distinguished from the original signer's signature (in polynomial time).
- **Identifiability**. The original signer can determine the identity of a proxy from his signature (if there are more proxy signers).
- **Undeniabilty.** A proxy signer cannot disavow his valid signature.

Detailed discussion may be found in [105].

Zhang in [167] noticed that the Mambo scheme does not provide

- **Nonrepudiation**. Neither the original nor the proxy signer can falsely deny later that he generated a signature.

In [167], the scheme from [105] has been enhanced to provide nonrepudiation.

The scheme in [105] allows the proxy to sign an arbitrary number of messages. Furthermore, using the warrant is not a systematic way to control the number of signed messages in electronic communication. In some situations, the signer may need to delegate his signature to the proxy for one-time/one-purpose only. For example, a manager may want, for security or administrative reasons, to restrict the proxy to signing on his behalf for one time only. Hence, a new type of proxy signature that is more restricted than the Mambo approach is needed. An efficient one-time signature can be used in this case. Kim *et al.* in [91] designed a one-time proxy signature using fail-stop signature to provide authentication to mobile agents applications. In our proposal, we use a class of one-time signatures, as described in Section 7.3, to design a new one-time proxy signature.

If we consider a "classical" type of scheme, where the original signer shares his secret keys with the proxy or generates new secret keys for the proxy, distinguishability and non-repudiation are not guaranteed, since both the original and the proxy signer know the secret keys. The character of a one-time signature allows us to adopt a principally new approach, where the secret keys are generated and kept by the proxy only, and the original signer has no share in the secret. The original signer only confirms the public keys and stores them with a trusted party (registry). The security properties such as unforgeability, proxy signer's deviation, verifiability, and undeniability of the scheme are the same as in the underlying one-time signature scheme $\mathcal{O}$. Introducing the proxy signer clearly does not cause any violation of these security properties unless the signature scheme is used more than once by an unreliable proxy. Signing several messages using the same set of keys reveals too many secret keys, and an eavesdropper could easily sign a false message using them. Our suggested solution involves the trusted party. Let us assume that, besides the public keys approved by the original signer, one more value will be stored by the proxy when signing a message. However, this action will be guarded by the trusted party and will not be allowed to be take place more than once. When signing the message m, the proxy will store there the value $h(m)$. In an additional step, the verifier will compute $h(m)$ for the received message m and check it against the value from the registry. Since this value can be stored to the registry just once, only one message can be legally signed. The scheme involves

a signer S, a proxy P, a verifier V, and a trusted party TP. It uses the one-time signature scheme $\mathcal{O} = (M, X, Y, h, v, \pi)$ where X is a sufficiently large set. In addition, we assume that h is extended to $h : X \cup M \to Y$ while still preserving its one-wayness.

Key generation

Signer S

1. asks P to start generating secret keys.

Proxy P

1. chooses v secret numbers : $s_1, s_2, \ldots, s_v \in X$
2. computes $p_1 = h(s_1), p_2 = h(s_2), \ldots, p_v = h(s_v)$
3. passes $(p_1, p_2, \ldots, p_v)$ to S.

Signer S

1. verifies that the p's are from P (a one-time signature of P can be used for signing the list of p's)
2. makes $(p_1, p_2, \ldots, p_v)$ public, registered to the name of S.

Signing

Proxy P

1. computes $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. computes $q = h(m)$ and registers this value with TP as a one-time writable value
3. sends $(m, s_{j_1}, \ldots, s_{j_r})$ to V.

Verification

Verifier V

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. computes $h_1 = h(s_{j_1}), h_2 = h(s_{j_2}), \ldots, h_r = h(s_{j_r})$
3. computes $q' = h(m)$
4. fetches $p_{j_1}, p_{j_2}, \ldots, p_{j_r}$ and q from TP
5. accepts the signature if and only if $h_1 = p_{j_1}, h_2 = p_{j_2}, \ldots, h_r = p_{j_r}$ and $q' = q$.

The proxy uses its private keys, and the public keys are stored with a trusted party; hence, the proxy cannot deny signing or revealing the secret to a third agent, which is a danger occurring in most of the established proxy signature schemes. Since the signer and the proxy do not share the same key, non-repudiation is achieved. Sending keys by the proxy to the signer in the key generation phase does not compromise security since these keys will become public in any case. The role of the original signer is to endorse the public-keys generated by the proxy signer to the registry. This step is crucial; otherwise, any agent may claim itself to be a proxy for the original signer.

7.5 A (t, n) threshold one-time signature scheme

A particularly interesting class of society-oriented cryptography is the threshold cryptographic, which deals with transformation of subsets of t or more participants from a group of n members. A digital signature is an integer issued by a signer which depends on both the signer's secret key and the message to be signed. In conventional cryptosystems, the signer is a single user. However, the process of signing may need to be shared by a group of participants. The first attempts at designing a shared signature were made by Boyd [27]. Threshold RSA [53] and Threshold ElGamal [101] signatures are examples of threshold multisignature schemes that require the presence of t participants of the group to sign a message. Both schemes exploit the threshold Shamir secret sharing method to generate shares of signatures.

Here, we attempt to expand the idea of threshold signatures from the conventional cryptosystems transformations into one-time signatures in order to benefit from its efficiency properties in speeding-up the verification process. Our model consists of a group of signers $S_i, i = 1, 2, \ldots, n$ and a verifier V. A one-time signature scheme $\mathcal{O} = (M, F, Y, h, v, \pi)$ is used, where F is a finite field and Y is a finite set, and both are sufficiently large. A threshold value $t \leq n$ is specified in advance. Not less than t signers are required to sign a message.

7.5.1 A scheme with a trusted party

In our first scenario, two more parties are involved: a trusted distributor D, and a trusted combiner C.

The idea behind this scheme is to let the distributor D choose the secret keys of the general one-time signature scheme of the group.

The shares of these secret keys for the particular signers are computed by D using

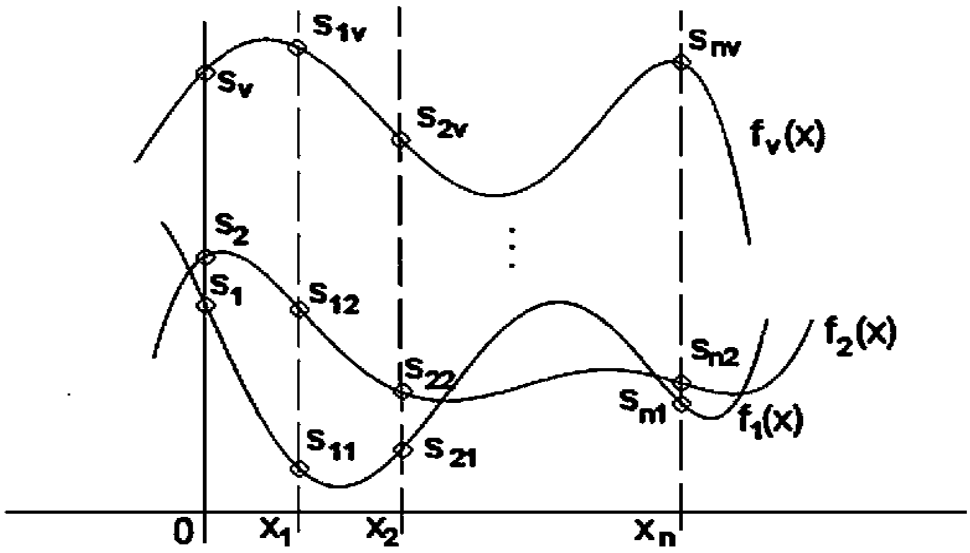

Figure 7.1: One-time signature polynomials interpolation

the Shamir secret sharing algorithm, and they are then distributed to the participants. For each secret key s_j, a distinct polynomial f_j of degree $t-1$ with $f_j(0) = s_j$ is used to create secret shares. A public value x_i is associated with each signer S_i; his secret share on the key s_j is then $s_{i,j} = f_j(x_i)$. The set of polynomials comprising the system is illustrated in Figure 7.1. Each intersection of a polynomial with the y axis represents a secret key. Two or more shares of the same signer may be identical, since several polynomials may have a common value in some of the points x_i (the graphs may intersect on a vertical line at x_i). This clearly does not compromise the security of the system, since this information is known only to D and to the signer. The secret keys s_j are chosen to be pairwise distinct; hence no two polynomial graphs intersect on the y axis.

For clarity, we detail the following Shamir threshold scheme, by which the dealer distributes the shares of the secret keys $s_1, s_2, \ldots$ among n participants. The combiner is then able to recover the signature, provided he knows t valid shares.

Key generation and share distribution

Distributor D

1. chooses randomly v pairwise distinct elements $s_j \in F, j = 1, \ldots, v$
2. computes the v values $p_j = h(s_j), j = 1, \ldots, v$ and makes them public (registered to the group name)
3. chooses randomly n pairwise distinct non-zero elements $x_i, i = 1, \ldots, n$ from F and makes them public
4. chooses randomly v polynomials $f_j(x) = f_{j,0} + f_{j,1}x + \cdots + f_{j,t-1}x^{t-1}$, $for j = 1, \ldots, v$, satisfying $f_j(0) = f_{j,0} = s_j$
5. computes $s_{i,j} = f_j(x_i), i = 1, \ldots, n, j = 1, \ldots, v$
6. sends $(s_{i,j})_{j=1,\ldots,v}$ by a secure channel to the signer $S_i, i = 1, \ldots, n$ (secret share for the partial signer S_i)

Signing

Signer $S_i, i \in \{i_1, i_2, \ldots, i_t\}$

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. sends the partial signature $(s_{i,j_1}, s_{i,j_2}, \ldots, s_{i,j_r})$ to C.

Combiner C

1. waits to receive partial signatures from (at least) t signers $S_{i_1}, \ldots, S_{i_t}$
2. using Lagrange interpolation, recovers the polynomials $f_{j_k}(x)$, $k = 1, \ldots, r$, based on the conditions $f_{j_k}(x_{i_1}) = s_{i_1,j_k}, \ldots, f_{j_k}(x_{i_t}) = s_{i_t,j_k}$
3. finds $(s_{j_1}, s_{j_2}, \ldots, s_{j_r}) = (f_{j_1,0}, f_{j_2,0}, \ldots, f_{j_r,0})$
4. sends $(m, s_{j_1}, s_{j_2}, \ldots, s_{j_r})$ to V.

Verification

Verifier V

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. fetches p_{j_k}, $k = 1, \ldots, r$ from the registry
3. checks whether $p_{j_k} = h(s_{j_k})$, $k = 1, \ldots, r$.

Since the usual Shamir perfect secret sharing is used, at least t signers are necessary to find any of the group secret keys. The fact that at most one message will be signed using the same signature may be guaranteed by the trusted combiner, so the multiple signature problem vanishes. In our scheme, the trusted distributor D knows all the secret keys; therefore, his reliability must be without doubt. The next version, without a trusted party, avoids such strict requirements. Observe that the combiner C knows only those secret keys which are used for the signature and which will be revealed to the verifier.

The computation of the shares involves nv times evaluation of a polynomial of degree $t-1$ by D and r times Lagrange interpolation of a polynomial of degree $t-1$ by C. In addition, D, V and each partial signer must compute $\pi(m)$ and D and V compute v and r values of the function h, respectively. The signers may compute π in parallel. It is worth noting that the verification of the one-time signature scheme is as efficient as without secret sharing.

7.5.2 A scheme without a trusted party

The scenario without a trusted party works the same way as the one with the trusted party; the steps of the distributor D and combiner C are performed by the signers themselves. In particular, the set of n signers is used as a parallel n-processor computer.

Key generation and share distribution

Signer $S_i, i = 1, \ldots, n$

1. chooses randomly a non-zero element $x_i \in F$ and makes (i, x_i) public
2. chooses randomly $s_j \in F$ (secret key) for each $j = 1, \ldots, v$ such that $(j-1) \bmod n + 1 = i$
3. computes the value $p_j = h(s_j)$ and makes the pair (j, p_j) public (registered to the group name)
4. chooses randomly a polynomial $f_j(x) = f_{j,0} + f_{j,1}x + \cdots + f_{j,t-1}x^{t-1}$ satisfying $f_j(0) = f_{j,0} = s_j$
5. computes $s_{i',j} = f_j(x_{i'}), i' = 1, \ldots, n$
6. sends $s_{i',j}$ by a secure channel to the signer $S_{i'}, i' = 1, \ldots, n$; $i' \neq i$ (secret share for the signer $S_{i'}$)

Signing

Signer $S_i, i \in \{i_1, i_2, \ldots, i_t\}$

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. sends the partial signature $(s_{i,j_1}, s_{i,j_2}, \ldots, s_{i,j_r})$: the triple (i, j_k, s_{i,j_k}) is sent to S_{i_q} where $q = (j_k - 1) \bmod t + 1, k = 1, 2, \ldots, r$
3. using Lagrange interpolation, based on the conditions $f_{j_k}(x_{i_1}) = s_{i_1,j_k}$, $f_{j_k}(x_{i_2}) = s_{i_2,j_k}, \ldots, f_{j_k}(x_{i_t}) = s_{i_t,j_k}$,recovers the polynomial $f_{j_k}(x)$ for each complete t-tuple $((i_1, j_k, s_{i_1,j_k}), \ldots, (i_t, j_k, s_{i_1,j_k}))$ received
4. for each polynomial f_j recovered, finds $s_j = f_{j,0}$
5. for each polynomial f_j recovered, sends (m, j, s_j) to V.

Verification

Verifier V

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. fetches p_{j_k}, $k = 1, \ldots, r$ from the registry
3. waits until all triples (m, j_k, s_{j_k}), $k = 1, \ldots, r$ have been received
4. checks whether $p_{j_k} = h(s_{j_k})$, $k = 1, \ldots, r$.

Let r_0 be the minimum length of a signature and v the total number of secret keys. In our scheme, each of the n signers generates $\lceil v/n \rceil$ secret keys. We claim that if $\lceil v/n \rceil (t-1) < r_0$, then at least t out of the n signers are necessary to create a valid signature. Indeed, $(t-1)$ signers know at most $\lceil v/n \rceil (t-1)$ secret keys. If the condition holds, this is not enough to create a signature of length r_0. On the other hand, the multiple signatures problem arises again here. Several messages may be signed even without malicious intention, since two independent subgroups of size t may sign two different messages. An improper solution of this problem may allow an existential forgery in the following way. After a valid signature of some message is created, a malicious agent (possibly identical with a subgroup of at most $t-1$ signers) may use some of the secret keys from the signature, in combination with the secret keys known to the subgroup of signers, to create a signature of a different message. This problem may again be resolved by using the "trusted registry" as in Section 7.4; the scheme and the proof of its security is provided in Section 7.6. We leave open the problem of designing a better scheme for one-time signatures that would solve this problem.

The complexity considerations from Part 7.5.1 are valid, except that, instead of the time necessary for computing nv polynomial values, the time for computing $\max(n, n\lceil v/n \rceil \approx v)$ values is required, since the signers may compute in parallel. In a similar way, only the time for $\lceil r/t \rceil$ Lagrange interpolations is necessary. How realistic is the condition $\lceil v/n \rceil (t-1) < r_0$? If the scheme of Lamport ([97]) is used to sign a message of length μ, then $v = 2\mu$ are generated, and the signature consists of μ keys. Our condition is satisfied if $t < n/2 + 1$.

7.6 A (t, n) threshold proxy one-time signature scheme

In this section, we combine the ideas from Section 7.5 and Section 7.4 and propose the following model. A group of n signers $S_i, i = 1, 2, \ldots, n$ wants to allow any subgroup of at least t signers to sign a message using a one-time signature. In our solution, the group will play the role of the original signer, who delegates his right to use a one-time signature to any subgroup of $t \leq n$ (proxy). The signature is to be verified by a verifier V. A one-time signature scheme $\mathcal{O} = (M, F, Y, h, v, \pi)$ with the security parameter K is used, where F is a finite field and Y is a finite set, both sufficiently large. Again, we assume that h is a one-way hash function, $h : M \cup F \to Y$. The trusted party TP is required only to keep the public keys and to prevent repeated signing. The start of the keys generation should be initiated in a suitable coordinated way.

Key generation and share distribution

Signer $S_i, i = 1, \ldots, n$

1. chooses randomly a non-zero element $x_i \in F$ and makes (i, x_i) public
2. chooses randomly $s_j \in F$ (secret key) for each $j = 1, \ldots, v$ such that $(j-1) \bmod n + 1 = i$
3. computes the value $p_j = h(s_j)$ and sends (j, p_j) to TP
4. chooses randomly a polynomial $f_j(x) = f_{j,0} + f_{j,1}x + \cdots + f_{j,t-1}x^{t-1}$ satisfying $f_j(0) = f_{j,0} = s_j$
5. computes $s_{i',j} = f_j(x_{i'}), i' = 1, \ldots, n$
6. sends $s_{i',j}$ by a secure channel to the signer $S_{i'}, i' = 1, \ldots, n, i' \neq i$ (secret share for the signer $S_{i'}$)

Trusted Party TP

1. verifies that each p_j is from a proper S_i (a one-time signature of S_i can be used for signing the pair (j, p_j))
2. makes $(p_1, p_2, \ldots, p_v)$ public, registered to the name of the group.

Signing

Signer $S_i, i \in \{i_1, i_2, \ldots, i_t\}$

1. finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. computes $q = h(m)$ and registers this value with TP; if q is different from a value already registered with TP, S_i stops signing
3. sends the partial signature $(s_{i,j_1}, s_{i,j_2}, \ldots, s_{i,j_r})$: the triple (i, j_k, s_{i,j_k}) is sent to S_{i_q} where $q = (j_k - 1) \bmod t + 1, k = 1, 2, \ldots, r$
4. using Lagrange interpolation, based on the conditions $f_{j_k}(x_{i_1}) = s_{i_1,j_k}$, $f_{j_k}(x_{i_2}) = s_{i_2,j_k}, \ldots, f_{j_k}(x_{i_t}) = s_{i_t,j_k}$, recovers the polynomial $f_{j_k}(x)$ for each complete t-tuple $((i_1, j_k, s_{i_1,j_k}), \ldots, (i_t, j_k, s_{i_1,j_k}))$ received.
5. for each polynomial f_j recovered, finds $s_j = f_{j,0}$
6. for each polynomial f_j recovered, sends (m, j, s_j) to V.

Verification

Verifier V

1. after receiving the first triple (m, j_k, s_{j_k}) finds $(j_1, j_2, \ldots, j_r) = \pi(m)$
2. computes $q' = h(m)$
3. fetches p_{j_k}, $k = 1, \ldots, r$ and q from TP
4. waits until all triples (m, j_k, s_{j_k}), $k = 1, \ldots, r$ are received
5. checks whether $p_{j_k} = h(s_{j_k})$, $k = 1, \ldots, r$ and $q' = q$.

As in Part 7.5.2, each of the signers knows at most $\lceil v/n \rceil$ secret keys of the group. Therefore, $(t-1)$ signers will not be enough to sign a message only if $\lceil v/n \rceil (t-1) < r_0$, where r_0 is the minimum signature length for messages under consideration. This fact is, expressed more formally in the following theorem.

Theorem 7.1 *Let Q be a polynomial. Let $\lceil v/n \rceil (t-1) < r_0$. Assume an adversary $\mathcal{A}$ who knows at most $\lceil v/n \rceil (t-1)$ secret keys and at most $t-1$ shares for any secret key. Then the probability that $\mathcal{A}$ will produce in time $Q(K)$ a valid signature for some message is less than $Q(K)/2^K$.*

Proof. Since $\lceil v/n \rceil (t-1) < r_0$, $\mathcal{A}$ has to find at least one additional value of a secret key. To do this in time $Q(K)$, $\mathcal{A}$ has to find either the share by breaking the perfect secret sharing algorithm of Shamir, or the one-way function h. Neither can be done with a probability greater than $Q(K)/2^K$. □

We do not assume in the theorem that $\mathcal{A}$ has a knowledge of at least one valid signature. If this is the case, then trying to forge a signature for another message makes no sense, since the verifiction will fail on the information kept by the TP. However, there is another problem connected to the involvement of the TP which we call a *blocking attack*. A malicious agent may send a fake hash of a message to the TP without an intention of signing the message. This will block the possibility of any valid signing by the group. This can be avoided by the following activity of the TP. In the key generation and share distribution part of the scheme, TP chooses a random value α_i for each signer S_i and sends it to S_i by a secret channel. In step 2 of the signing part, each S_i sends the pair $h(m), h(h(m)||\alpha_i)$ to the TP. When the TP receives t such

pairs, she registers the value $h(m)$. The existence of TP is therefore essential as part of a solution to the problem, and we leave it as an open problem to find a simpler solution not involving TP in the computations.

7.6.1 Special case: $t = 1$

A particular case of interest in this scheme is when $t = 1$, which depicts the *anycast* model. The anycast authentication problem will be discussed in detail in the chapter. Briefly, the anycast model represents the situation where any of a group of n servers (signers) may provide the same (equivalent) service to a client (verifier). The method of nominating the actual signer is an optimization problem, and it is done by the network infrastructure based on a number of criteria such as less communication overhead or more available memory, and so on. In the solution, an additional party (a group coordinator) may behave as the original signer in our $(1, n)$-threshold scheme while the servers behave as the proxies. The original signer delegates his power to n proxy signers and the verifier verifies a message from "one" of the proxy signers. Although the above $(1, n)$- threshold scheme of one-time signature is theoretically correct, it is not of practical concern since the signer needs to generate different secret keys for each proxy correspondence.

7.7 Summary

One-time signatures was already used in other studies of multicast security as an efficient tool of authentication (e.g., [133]). However, it was not been used in the context of group-oriented cryptography such as in threshold or proxy scenarios. In these scenarios, typical digital signatures schemes were usually in the picture. With the need for efficient society-oriented methods, one-time signatures were the potential to fill this gap. In this chapter, we have proposed several schemes related to authentication with one-time signature. The first case deals with the implementation of a one-time signature in proxy delegation; the second shows how to use a (t, n) threshold one-time signature in a group of signers, and the third scheme combines the above two ideas into a (t, n) threshold proxy one-time signature. An extension to this work, left as an open problem, is to design a one-time signature scheme to prevent multiple message signing using the same set of one-time signature keys.

Chapter 8

Authentication of Anycast Communication

Anycast is a communication mode in which the same address is assigned to a group of servers and a request sent for a service is routed to the "best" server. The measure of best could be the number of hops, available bandwidth, load of the server, or any other measure. With this scenario, any host could advertise itself as an anycast server in order to launch a denial-of-service attack or provide false information. In this chapter, we solve this problem by proposing an authentication scheme for anycast communication.

8.1 Introduction

The Internet is increasingly being viewed as providing services rather than just connectivity. As this view has become more prevalent, the important considerations in the provision of such services are reliability and availability of the services to meet the demands of a large number of users; this is often referred to as scalability of the service. There are many approaches to improving the scalability of a service, but the common one is to replicate the servers. Server replication is the key approach for maintaining user-perceived quality of service within a geographically widespread network. This is empowered by the underlining network infrastructure known as *anycast communication*. The anycasting communication paradigm is designed to support server replication by allowing applications to easily select and communicate with the best server, according to some performance policy criteria, in a group of content-equivalent servers.

With regard to the above description of anycast communication, the system can be subject to a number of security threats. In general, there are at least two security issues in anycasting, which are mainly related to authentication. First, it is clear that malevolent hosts could volunteer to serve an anycast address and divert anycast datagrams from legitimate servers to themselves. Second, eavesdropping hosts could reply to anycast queries with inaccurate information. Since there is no way to verify

membership in an anycast address, there is no way to detect that the eavesdropping host is not serving the anycast address to which the original query was sent. In both scenarios, the security requirement is anycast server authenticity.

In this chapter, we consider the problem of authentication in anycast communication, and we propose an authentication solution which is closely related to the concept of *proxy signatures*. In the next section, we review related work in the area of proxy signatures, and then we describe the anycast model. Next, we describe the proposed scheme and discuss the security and performance issues of the scheme. Finally, we conclude with some remarks.

8.2 Related work

Delegation of rights is a common practice in the real world. A manager of an institution may delegate to one of his deputies the capability to sign on behalf of the institution while he is on holiday. For electronic transactions, a similar approach is needed to delegate the manager's digital signature to the deputy.

Proxy signature is a signature scheme where an original signer delegates his/her signing capability to a proxy signer, and then the proxy signer creates a signature on behalf of the original signer. When a receiver verifies a proxy signature, he verifies both the signature itself and the original signer's delegation. Mambo, Usuda and Okamoto (MUO) [105] were the first to introduce the concept of proxy signature. They gave various constructions of proxy signature schemes and their security analysis [105].

Lee *et al* [99] noticed that the MUO proxy scheme does not satisfy the strong undeniability property, that is, a proxy signer can repudiate the fact that he has created the signature. Based on this weakness, they classified proxy signature schemes into strong and weak ones according to their undeniability property. Another important criterion of classification is whether the proxy signature is *proxy-protected* or *proxy-unprotected*. If it is proxy-protected, the original signer cannot first create or know the proxy secret key, and therefore it cannot create a proxy signature. Other related types of signatures are the multiproxy signature scheme [84] and the threshold scheme [167]. In this sort of signatures, there is a set (group) of proxies, rather than a single proxy, collaborating to generate a proxy signature.

8.2.1 The Schnorr signature

Let p, q be primes such that q divides $(p-1)$ and $q \geq 2^k$ where k is the security parameter. Let $g \in Z_p^*$ and let H be a hash function with values in Z_p. The Schnorr signature scheme ([137]) can be described as follows:

Initialization

Signer

1. Chooses the secret key $x \in Z_p^*$
2. Computes the public key
   $y = g^{-x} \pmod{p}$.

Signing of a message m

Signer

1. Chooses a random $K \in Z_q^*$.
2. Computes $r = g^K \bmod p$.
3. Computes $e = H(m, r) \bmod q$.
4. Computes $s = K + ex \bmod q$.
5. Sends (m, r, s) to the client.

Verification of the signature (r, s) **of** m

Verifier

1. Computes $e = H(m, r) \bmod q$.
2. Checks whether $r = g^s y^e \pmod{p}$.
3. Accepts if the check is OK; otherwise, REJECT.

8.2.2 Schnorr-based proxy signatures

Proxy signature schemes could be designed using any standard and secure signature scheme. However, because the Schnorr scheme proved to be an elegant and secure signature scheme in the random oracle model [126], a number of proxy signature schemes that use the discrete logarithmic problem apply the Schnorr scheme as the standard signature algorithm.

Boldyreva, Palacio, and Warinschi in [25] proposed a provably secure scheme called the Triple-Schnorr proxy signature scheme, which is modified from the [92] scheme, for warrant-based delegation. They also presented a formal definition and security notion for their proxy signature.

Lee, Kim and Kim (LKK) in [99] proposed a Schnorr warrant-based proxy signature

scheme (see Section 7.4 for types of delegations and security requirements). The same authors in [98] claimed that their Schnorr-based LKK scheme is as secure as the Schnorr scheme and satisfies all the security requirements. Wang *et al* in [161] prove that the schemes in [99, 98] are insecure, and provide a security analysis to their work.

8.3 Anycast Scenario

Anycast addressing has become a part of the IPv6 new generation internet protocol ([83]). In anycast communication, a common IP address (*anycast address)* is used to define a group of servers that provide the same service. A client sender desiring to communicate with only one of the servers sends datagrams with the IP anycast address. The datagram is routed using an anycast-enabled router to the best server of the group. Recall that the best server is elected based on a criterion such as the minimum number of hops, more available bandwidth, or least load on the server. Anycasting simplifies the task of finding an appropriate server considerably. Users, instead of manually consulting a list of servers and choosing the best one, can be connected to the best server automatically. The client does not care which of the servers is assigned to him for the communication. In fact, various servers may participate in the different parts of one communication session.

It is worth mentioning that the concept of anycasting is related to multicasting. While multicasting involves building and maintaining a distribution tree from a single server to multiple clients, anycasting involves the concept of redirecting the client to a server from a group of servers.

8.3.1 The Model

The model for anycast communication as depicted in Figure 8.1 consists of a group of **anycast servers** $A_1, A_2, \ldots, A_n$, and a **client** C. We introduce an authentication scheme based on an additional agent called a **group coordinator** G. The group coordinator is the main player in the setting and is considered as the *original signer*. The group coordinator is considered to have the power to distribute the signature rights for the whole anycast group. She delegates her rights to all servers in the group, and therefore it is used to prevent malicious hosts from pretending that they are the anycast group members.

The communication in the model consists of two phases:

1. *Initialization.* The communication of each server with the group coordinator. A signature delegation algorithm is used in this communication. Each server starts playing the role of the coordinator's proxy.

2. *The actual serving.* The anycast server uses the delegated signature, together with the proof of his delegation.

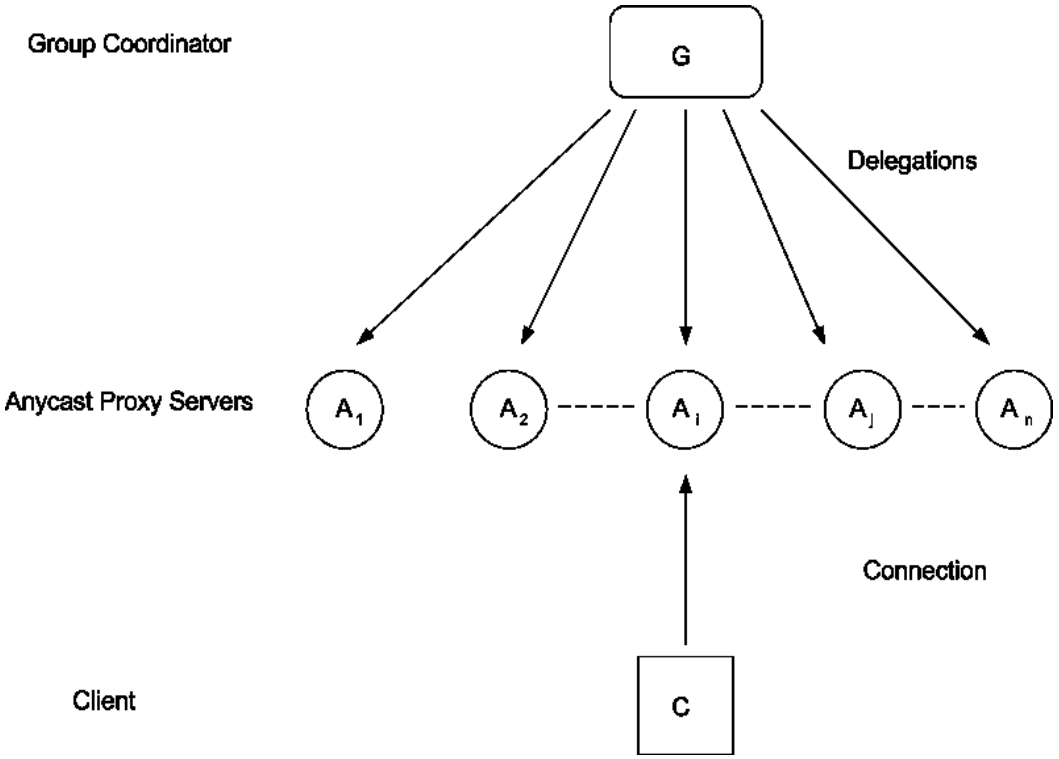

Figure 8.1: Anycast Model

In this model, we assume that all communications between players in the model are done in public (i.e. insecure) channels. We assume that the communication from the servers to the client is based on the authentication of the servers by a suitable signature scheme. We also assume that the security attacks may be launched *externally* by an adversary, or *internally* by any of the anycast players in the model.

In our solution, described in the next section, we will apply a Schnorr-based digital signature scheme (as described in [138]), which was obtained by improving the scheme from [167].

In summary, the key characteristics of the model are:

1. it is applied to multiple proxies,
2. each proxy has the delegation, but each message is signed by one proxy only,
3. it uses a partial delegation mode,
4. it assumes the existence of a trusted third party to solve possible disputes,
5. it uses insecure public communication channels,
6. it assumes that all messages exchanged between the coordinator and any of the anycast servers during key exchange protocol are authenticated. The authentication should include the identities of both communication parties.

8.3.2 Security Requirements

The security requirement for an anycast model is similar to the requirement of a typical proxy signature which was first specified in [105] and already mentioned in Section 7.4. It basically inherits the security properties of the original Schnorr signature scheme and of the signature delegation scheme ([167].

For our model, we list these requirements to fit the anycast scenario settings:

R1) *Verifiability:* From the proxy signature, a verifier can be convinced of the group coordinator's agreement to the signed message.

R2) *Strong Unforgeability.* Only the group coordinator can authorize a new anycast server. In particular, an existing anycast server cannot do the same.

R3) *Identifiability.* The identity of an anycast server A_i can be determined from the server's signature.

R4) *Strong Undeniability.* The server A_i cannot deny that he (or someone to whom he revealed his secret) is the author of the signature.

R5) *Nonrepudiability:* The group coordinator should not compute a valid proxy key pair under the name of the proxy signer.

8.4 The Anycast Scheme

In our anycast scheme, the group coordinator G will play the role of the original signer, which delegates his signature rights to all the members of the anycast group which act as proxy signers. His public key y will be the public key of the whole group of anycast servers. For this delegation, we propose using the nonrepudiable proxy signature scheme from [167] based on the scheme from [105].

Assume a group of **anycast servers** $A_1, A_2, \ldots, A_n$, a **client** C and a **group coordinator** G. Let p, q be large primes such that q divides $(p-1)$ and let $g \in Z_p^* = GF(p)$. Let M be the set of messages (not necessarily of uniform length) and $H : M \rightarrow Z_p$ be a hash function.

In the initialization part, we assume that the group coordinator G has already chosen his secret key x, and public key $y = g^x \bmod p$ is registered with the trusted party.

Initialization

Group Coordinator G

1. For each $1 \leq i \leq n$,
   chooses a random value $z_i \in Z_q$,
   computes $u_i = g^{z_i} \bmod p$ and
   sends u_i to the i-th proxy-server.

3. Computes $v_i = t_i x + z_i \pmod q$.
4. Sends v_i to the i-th Proxy server.

Server A_i $(i = 1, \ldots, n)$

2. Repeatedly selects a random
   value $\alpha_i \in Z_q$ until the value
   $t_i = g^{\alpha_i} u_i \bmod p$ belongs to Z_q^*,
   and sends t_i to the Coordinator.

5. Computes $x_{P_i} = v_i + \alpha_i \pmod q$
6. Checks the equality
   $g^{x_{P_i}} = y^{t_i} t_i \bmod p$.
7. Accepts x_{P_i} as her secret proxy key.
8. Accepts $y_i = y^{t_i} t_i \bmod p$
   as her public proxy key

Signing of a message m and verification of the signature

Server A_i

1. Chooses a random $K \in Z_q^*$.
2. Computes $r = g^K \bmod p$.
3. Computes $e = H(m, r)$.
4. Computes $s = K + e x_P \bmod q$
5. Sends (m, r, s, t_i) to the client.

Client C

6. Computes $e = H(m, r)$.
7. Verifies
   $r = g^s (y^{t_i} t_i)^{-e} \pmod p$.
8. Accepts; if the check is OK,
   otherwise, REJECT.

8.4.1 Security Issues

We can identify a number of possible forgery attacks against our anycast model which can have different strengths and can be used in different settings. The security threats to the anycast model include attacks similar to those existing for proxy signature schemes, as well as extra attacks possible due to the existence of multiple proxy servers in the anycast group.

We examine the following attacks in the anycast model.

T1) *Existential Forgery:* In this attack, an adversary tries to forge a proxy signature. He outputs a valid proxy signature σ for a message m, which was not signed by a proxy signer. The security of this attack is measured by the difficulty of producing an existential forgery under adaptive chosen-message attack. The difficulty of forging the server's signatures follows from the properties of the underlying Schnorr signature scheme, and it can be reduced to the difficulty of solving the discrete logarithm problem (DLP) which has been proven to be secure in the random oracle model [126]. The existence of this attack violates the security requirements R2, R3, and R4.

T2) *Dishonest group coordinator:* The group coordinator produces a valid proxy signature which looks as if it was generated by a proxy. Formally, if G wants to sign a message pretending to be A_i, he has to use the public key

$$y_p = y^{t_i} t_i = g^{x_i}$$

that is, he uses x_i as the actual key. To find x_i, he needs to solve the equations:

$$g^{x_p} = y^{t_i} t_i$$

$$g^{x_p} = g^{x t_i} g^{\alpha_i} g^{z_i}$$

$$g^{x_p - x t_i - t_i} = g^{\alpha_i}$$

Finding x_p is equivalent to finding $x_p - x t_i - t_i$ since $x t_i$ and t_i are known to G. This is equivalent to DLP because α_i is a random number chosen by the proxy. The existence of this attack violates the security requirements R4 and R5.

T3) *Impersonating Proxy:* In this scenario, we assume a fake proxy $\bar{A}$ not designated as anycast proxy signer by G. $\bar{A}$ pretends to be an anycast proxy signer by forging a valid proxy key pair without the coordinator's agreement. However, this attack is difficult since the knowledge of the secret key x of the original signer is necessary to create a secret proxy key for the new server. Finding a suitable value of $\bar{t}_i$ to be used in the verification algorithm is as difficult as finding the value of x, and basically requires solving the discrete logarithm problem. The existence of this attack violates the security requirement R2.

T4) *Dishonest proxy signer:* Here, a dishonest proxy signer can cheat to get a signature generated by the original signer on any message. As described in [100] this attack is possible on any scheme belonging to some variants of ElGamal-type signature (including Schnorr signature). For this attack to take place during the proxy key generation phase is indeed possible. However, as illustrated in [65], a cheating attack is successful only if it is not detectable. They showed that the original signer can prove that the proxy signer was cheating in the key delegation protocol. In our anycast scheme, G is not using his pair of keys (x, y) for signing, so forging the signature does not make sense. However, the keys (x, y) may be misused as proxy keys with the value $t_i = 1$. This can be easily prevented by not allowing the use of $t_i = 1$ in the verification scheme. The existence of this attack violates the security requirement R1.

T5) *Transferring Attack:* In this scenario, an adversary converts the proxy signature σ into a new one $\bar{\sigma}$, where $\bar{\sigma}$ is also a valid proxy signature, but on behalf of a different original signer. Note that, in our scheme, the proxy signer can be identified based solely on the value t_i, where G must have a record of the proxy authentication of t_i. The attacker does not have such a valid record if the authentication includes the identification of both G and A_i. The existence of this attack violates the security requirements R2.

T6) *Colluding attack:* In this attack, the adversary's goal is to reveal the secret key x of the original signer by getting available information from some (or all) proxy servers of the anycast group. This approach does not seem to have a greater chance of success than getting multiple signatures from a single proxy in the

underlying signature delegation scheme. This follows from the fact that distinct random values z_i and α_i are used to generate the secret proxy keys x_{P_i} for distinct anycast servers. The existence of this attack violates the security requirements R2.

A remark may be raised with regard to the security requirement R3. Note that we have not required a strong identification as is required in other proxy schemes. This is because the identification of A_i by the client C is not desirable in anycast applications. The client should deal with the anycast servers group seemingly as if dealing with a single server. So, from the client's (verifier's) perspective, it is not vital to know by which server she has been served (signed). However, in case of a dispute, the identity of the anycast signer can be revealed from the value t_i, with the cooperation of G, who has the authentication of A_i on t_i.

Warrant-based vs. Partial-delegation

Using partial delegation, as in our scheme, provides sufficient security services to our anycast application and with less computational overhead. However, delegation by warrant provides more security services, but with extra computational overhead involved. In fact, this is the dilemma between efficiency and security between warrant-based and partial-delegation proxy signatures. While it was possible to provide a rigorous security proof to warrant-based delegation protocols [25], it was difficult to find a such to partial-delegation based proxy signature protocols. This is the reason behind the insecurity of most partial-delegation based schemes (see [161]). The provably secure Triple-Schnorr proxy signature scheme by Boldyreva *et al* in [25] could be, however, applied in our anycast scheme, but it would not be an efficient authentication solution to anycast communication. The choice between using warrant-based or partial-delegation sort of delegation (from security perspective) is 'application-specific'. One of the most necessary features of authenticating group-based applications is the efficiency. In our opinion, partial-delegation proxy-signatures protocols could be used securely, but with careful selected security requirements.

8.4.2 Performance Issues

Considering the performance issues, the initialization phase requires, besides several multiplications, at least four exponentiation $mod\ p$ to create a secret key for one server.

However, this is done just once, and thus this part of the computational complexity need not be of great concern. Signing of a message by a server requires one exponentiation, and verification by the client requires three exponentiations. However, in the frequent case where a single server sends several messages to the same client, the value $y^{t_i}t_i$ can be kept in the client's memory and need not be computed repeatedly.

In comparison with delegation by warrant, partial delegation is far less computationally expensive. In warrant-based schemes, the original signer needs to sign the warrant in roughly the same amount of time required for standard signing, but the verifier requires twice the time to verify a standard signature. In a discrete logarithm based scheme such as the Schnorr signature scheme, a proxy signature requires four modular exponentiations, while it needs two normal signatures verification when used with partial delegation. Yet, because of the nature of anycast application, it is not recommended to impose warrants on servers for delegation and verification; otherwise, it would increase communication overhead and turn out to be impractical.

8.5 Summary

Anycast is a new communication mode in the context of group communication in which a server is selected from a group of servers based on some criterion such as the greater bandwidth available to the server or the less utilized server. It is the defacto of multicast communication. The anycast paradigm is usually used in server replication environments to provide service availablity. Anycast become part of the IP version 6 suite. The security of anycast communication has been discussed here, and we have focused on the authentication aspects since any server can pretend to belong to the anycast group. Our solution is a variant of the proxy signature scheme. An extension of this work may investigate the possibility of authenticating *distributed anycast servers* over different environments of different authentication methods.

Chapter 9

Authentication of Joining Operation

IP multicast is a dynamic and scalable communication mode. Any host can join a multicast group, by using a group membership protocol IGMP, without its identification information being released or known to other nodes. From the perspective of extendability, this is an attractive feature, but from a security perspective it represents a shortcoming. In this chapter, the secret sharing technique is used to construct two variants of signature schemes. The first scheme proposes a designated verifier signature to provide authenticity to a specified verifier. To provide both authenticity and privacy, a signcryption scheme (based on the same idea) is then introduced.

9.1 Introduction

A simple and at the same time vital operation in multicast communication is when a host uses a membership protocol such as Internet Group Membership Protocol (IGMP) to join a multicast environment and to become a member of the group. Based on a successful membership, the host is able to send or receive messages from other members of the group. The joining operation is a dynamic feature which allows a host to join and leave a group at any time without notice.

On one hand, in the current IP multicast models, the source of the multicast data is never aware of the identities of the receivers of multicast data. Although some applications may not need membership identification information, such as public information transmissions, others (for example subscription services and conferencing applications) may require precise information about receivership of the multicast group.

On the other hand, the current standards of IGMP protocol (version 1 and 2) [46, 59] do not incorporate security features. Other application layer authentication protocols are either not efficient or do not provide sufficient security services [73], and version 3 is not yet standardized. In either case, authenticating the source of the communication is an essential requirement in the joining operation.

In fact, the joining operation of the multicast is the easiest and the most vulnerable to security threats of all other phases of multicast communication which have been discussed earlier. This is because it is the first gateway for intruders on launch their attacks to a multicast system. The communication mode in the previous phases used to be 'internal' between the communication nodes and hidden from typical users. Also, launching an attack demands great expertise and professionalism. However, in joining the operation, the user has direct access to the multicast environment, and this is therefore of great security concern. Hence, it is not only the authentication that is a concern, but also the confidentiality, and the integrity of the exchanged information is an additional possible essential requirement. This integration of security services in a protocol is essential for modern security protocols.

Because of this necessity of securing the communication between the host and the group manager, we propose the following two protocols. The first protocol proposes a *designated verifier signature* (DVS) to provide peer authentication between a host and group manager. In other words, a host can convince one and only one specified recipient (the group manager) that it is legitimate user. However, unlike standard digital signatures, nobody else can be convinced about their validity or invalidity, nor does it not provide the non-repudiation property of traditional digital signatures. The idea then is extended to a second protocol known as *signcryption* which provide not only authenticity, but also confidentiality and integrity. The two cryptographic protocols are based on the concept of verifiable secret sharing. Section 9.3 illustrates how to generate a signature from the Shamir secret sharing method.

9.2 Related Work

The main contribution of digital signatures has been to provide authentication based on public-key cryptography. This was an essential requirement in many applications for many years. However, the different needs of digital signatures in the diverse and modern applications have required special variations to fulfill the stringent requirements imposed by these applications.

For example, typical digital signatures such as RSA or ElGamal (Section 2.5) provide authentication with non-repudiation. In this sort of signatures, anyone having a copy of the signature can check its validity. However, this self-authentication property is not always desirable, for example in some scenarios where personal private information such as medical records, tax information, or personal transactions, should not be

exposed to the public. It is intended for verification for a specific verifier only.

This variation of digital signatures is called *designated verifier signatures* (**DVS**) and it was firstly proposed by Jakobson, Sako, and Impagliazzo [85], and independently by Chaum [34]. Vergnaud and Laguillaumie in [158] provide a new scheme with a formal security model. The DVS are intended for a specific and unique designated verifier: the only one who can check their validity. Such signatures have numerous applications. For example, in group communication, a group (multicast) manager requires, for security reasons, to control the joining of users into a multicast domain. Therefore, a protocol that authenticates the joining of a particular host to a group is required. The verification is performed by a one-and-only-one entity, that is, the group manager. Calls for tenders, electronic voting, electronic auction or distributed contract signing are other examples. It can be noticed that a very efficient way to produce DVS is to use MAC or HMAC under the symmetric-key cryptography. Therefore, prior exchange of the secret keys is required in secure channels. A DVS scheme is called asymmetric if it uses a public-key cryptosystem. DVS in general do not provide non-repudiation, as is the case in any typical digital signature. Later, Vergnaud and Laguillaumie in [157] generalized the idea of DVS into multi-designated verifiers, where more than one verifier can be designated for verification.

DVS was one possible variation to digital signatures in terms of restricting the verifiers. Another variation to digital signature in terms of extending the functionality is required. Typical digital signatures provide authenticity but do not provide confidentiality of information, whereas in some applications, both confidentiality and authenticity are required. Therefore, another variation of digital signatures is needed to provide both privacy and authenticity. This sort of digital signature is called *signcryption*, and it was first introduced by Zheng [169] at Crypto 1997. For example, stock exchange communication requires authentication of the source of multicast as well as encryption of the private data.

9.3 Theoretical Description and Construction

The main structure of our system is based on the Shamir threshold secret sharing method described in Section 2.7. The main idea relates to the concept of verifiable secret sharing, illustrated in Section 2.7.3.

Shamir's (t, n) threshold scheme $(t \leq n)$ is a method by which a trusted party computes secret shares s_i, $1 \leq i \leq n$ from initial secret s, such that any t or more users

who pool their shares may easily recover s, but any group knowing only $t-1$ or fewer shares may not. Shamir secret sharing is based on Lagrange polynomial interpolation, and on the fact that a univariate polynomial $y = f(x)$ of degree $t-1$ is uniquely defined by t points (x_i, y_i) with distinct x_i (since these define t linearly independent equations in t unknowns). All calculations are done in $GF(p)$ where the prime p is selected to satisfy the security requirements. The scheme is constructed by a trusted party.

Since Shamir secret sharing attempts to create a unique polynomial which passes through a number of points, it can be exploited to produce a signature for a message. When the points (or shares later) were designed to be the signer and the verifier keys, then a mutual correlation is established and peer-to-peer authentication is possible.

The main idea of our construction is to treat the message that needs to be authenticated as a secret, as in a secret sharing scheme, and then to generate shares of the secret. The challenge for the verifier is to reconstruct the secret based on the knowledge of the shares, much as any secret sharing technique. If the verifier is able to compute the secret based on valid shares, this proves that the message is genuine since mathematically only unique points (shares) can reconstruct the secret. Any difference in the points would not reconstruct the message (secret). The selection or generation of shares should be difficult in order to prevent the adversary from forging the signature.

The main components of the construction are (3,3) Shamir secret sharing scheme, an intractable hash function H, and the Diffie-Hellman key-agreement cryptosystem [56]. The hash function is used as a one-way function. The Diffie-Hellman cryptosystem is used to establish a shared secret key over public channel. The public keys of the signer and the verifier are used as shares in the Shamir secret sharing system, and at the same time they provide strong authentication and mutual correspondence between the two communication parties.

Suppose two parties A and B want to establish an authentication relationship using their key pairs $(sk_A, pk_A), (sk_B, pk_B)$ respectively, where sk is the secret key and pk is the public key. Then, three points of the following shape need to be created:

$$\alpha = pk_B^{sk_A}\,, \qquad \beta = pk_A\,, \qquad s = H(m||\alpha)$$

where

$$f(0) = s, f(1) = \alpha, f(2) = \beta$$

The signature of the message m is $\sigma = f(3)$.

This means that the triplet α, β and σ are the shares of (3,3) Shamir scheme with the secret s. Hence, the system is constructed from three linear equations in three

unknowns, and therefore, using the Lagrange interpolation function, it is possible to determine a unique polynomial $f(x)$ of three parameters and of degree two:

$$f(x) = f_0 + f_1 x + f_2 x^2$$

In a protocol, A, the sender sends the message m and its signature σ to the verifier B. The verifier collects the authentic value of β from the public registry and computes the value α and the hash value of the received message. Then it takes the triplet α, β and σ and computes the secret s'. The verification of the signature is successful if $s = s'$; otherwise, it is rejected.

Note that the generated signature from the above formula is deterministic. To make it non-deterministic (or probabilistic), the parameter α can be raised to a random number, say $t \in [1, q-1]$, for prime q.

The use of the Diffie-Hellman cryptosystem has two benefits. First, it establishes a relationship between the signer and verifier. Second, it simplifies the task of exchanging the parameters of the interpolation polynomial between signer and verifier. Each of the principals can compute the parameters without extra communication overhead. Note also that the signature is only verified by the corresponding peer which has been designated by the sender.

9.4 Designated Verifier Signature

The concept of DVS was explained in the first two sections of this chapter. We gave a preamble with its theoretic construction in the previous section. In this section, we define the formal security model of our DVS scheme.

9.4.1 Formal Definition of DVS scheme

Definition 9.1 (Designated Verifier Signature Scheme). *Given an (interactive probabilistic Turing machines) entities A and B, and an integer k, an asymmetric DVS with security parameter k is defined by the following:*

- a **setup algorithm** (Setup): *it is a probabilistic algorithm which takes as input a security parameter k and outputs the public parameters pc,*
- a **key generation algorithm for signer** (SKeyGen): *it is a probabilistic algorithm which takes as input the public parameters pc and an entity A, and outputs a pair of secret/public keys (pk_A, sk_A),*

- a **key generation algorithm for verifier** (VKeyGen): *it is a probabilistic algorithm which takes as input the public parameters pc and an entity B, and outputs a pair of secret/public keys (sk_B, pk_B),*

- **designated verifier signing algorithm** (Sign): *it takes as input a message m, a secret key sk_A, and the public key of the verifier pk_B, and outputs designated verifier signature σ of m. This algorithm could be probabilistic or deterministic,*

- **designated verification algorithm** (Verify) *it takes as input a designated verifier signature σ, a message m, a public key pk_A, and secret key sk_B, and tests whether σ is a valid designated verifier signature of m with respect to the keys. It returns $\perp$ if the signature is invalid. This is a deterministic function.*

In brief, DVS = (Setup, SKeyGen, VKeyGen, Sign, Verify)

9.4.2 Security Notions

Attacks against digital signature schemes can be defined according to the goals of the adversary and to the resources that it can use. The strongest security notion, defined by Goldwasser, Micali and Rivest in [66], is known as *existential forgery against adaptive chosen message attack* (EF-CMA)[1]. An adversary $\mathcal{A}$, given the public key of the verifier B, as well as access to the hash function $\mathcal{H}$, to a signing oracle Σ and to a verification oracle Υ is allowed to query the verification oracle on any couple message/signature of its choice. In the adversary answer, there is a natural restriction that the returned message/signature has not been obtained from the signing oracle. This is because, if the adversary can return a signature obtained from the signing oracle, then this signature is by definition valid, and the forge is trivial. This is the natural restriction of all the signature schemes.

Security against existential forgery under chosen message attack.
Let $\mathcal{A}$ be EF-CMA-adversary. We consider the following experiment:

[1] By 'strong', we mean the difficulty and the vitality of the attack

$$\boxed{\text{Experiment } \mathbf{Exp}^{\mathsf{ef-cma}}_{DVS}(k)}$$
$$pc \leftarrow DVS.Setup(k)$$
$$(pk_B, sk_B) \leftarrow DVS.VkeyGen(pc, B)$$
$$(pk_A, sk_A) \leftarrow DVS.SKeyGen(pc, A)$$
$$(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{H},\Sigma,\Upsilon}(pk_A, pk_B)$$
$$\text{Return } \mathsf{DVS.Verify}(pc, m, \sigma, pk_A, sk_B)$$

where $\mathcal{A}$ has access to the oracles $\mathcal{H}, \Sigma, \Upsilon$, and to the keys pk_A, pk_B. $\mathcal{A}$ is allowed to query the signing oracle on the challenge message m, but is supposed to output a signature of the message m not given by Σ. We define the success of the adversary $\mathcal{A}$ to implement the existential forgery under chosen message attack as follows:

$$\mathbf{Succ}^{\mathsf{ef-cma}}_{DVS,\mathcal{A}}(k) = Pr[Exp^{\mathsf{ef-cma}}_{DVS,\mathcal{A}} = 1]$$

Security assumptions for Diffie-Hellman (DH) problems

An interesting new class of computational problems is called gap problems [119]. A gap problem is a coupling of inverting and decisional problems. More precisely, this problem is to solve an inverting problem with the help of an oracle for a decisional problem. The following are formal definitions and assumptions [158]:

Let G be a group of prime order q, and g be a generator of G. We define the following problems:

- **Computational Diffie-Hellman (CDH)**: let a and b be two integers smaller than q. Given g^a, g^b, compute g^{ab}.
- **Decisional Diffie-Hellman (DDH)**: let a, b and c be three integers smaller than q. Given g^a, g^b, g^c, decide whether $c = ab \ mod \ q$.
- **Gap Diffie-Hellman (GDH)**: let a and b be two integers smaller than q. Given g^a, g^b, compute g^{ab} with the help of a DDH oracle.

Definition 9.2 Prime-order-DH-parameter-generator.
A prime-order-DH-parameter-generator is a probabilistic algorithm that on security parameter input k outputs a triple (q,g,G) satisfying the following conditions: q is a prime with $2^{k-1} < q < 2^k$, *G is a group of order q, and g generates G.*

GDH. Let *Gen* be a prime-order-DH-parameter-generator. Let $\mathcal{D}$ be an adversary that takes on input $(X, Y) \in G^2$ and returns an element of $Z \in G$. We consider the following random experiments, where k is a security parameter:

$$\boxed{\text{Experiment } \mathbf{Exp}^{\mathsf{gdh}}_{Gen,\mathcal{D}}(k)}$$
$$(q, g, G) \leftarrow Gen(k)$$
$$\mathsf{setup} \leftarrow (q, g, G)$$
$$x \leftarrow [0, q-1], X \leftarrow g^x$$
$$y \leftarrow [0, q-1], Y \leftarrow g^y$$
$$Z \leftarrow \mathcal{D}(setup, X, Y)^{\mathcal{O}_{DDH}}$$
$$\text{Return 1 if } Z = g^{xy},$$
$$0 \text{ otherwise}$$

We define the corresponding *success* of $\mathcal{D}$ in solving the *GDH* problem

$$\mathbf{Succ}^{\mathsf{gdh}}_{Gen,\mathcal{D}}(k) = Pr[\mathbf{Exp}^{\mathsf{gdh}}_{Gen,\mathcal{D}}(k) = 1]$$

Definition 9.3 (GDH assumption) *Let $t \in N$ and $\varepsilon \in]0,1[$. GDH is said to be (t,ε)-secure if no adversary $\mathcal{D}$ running in time t has success $\mathbf{Succ}^{gdh}_{Gen,\mathcal{D}}(k) \geq \varepsilon$.*

Definition 9.4 (Security against existential forgery). *Let B an entity, k and t be integers and ε be a real in]0,1[. A Designated Verifier Signature DVS with security parameter k is said to be (t,ε) EF-CMA secure if no adversary $\mathcal{A}$ running in time t has a success $\mathbf{Succ}^{\mathsf{ef-cma}}_{DVS,\mathcal{A}}(k) \geq \varepsilon$.*

9.4.3 The DVS Scheme

The scheme consists of three parts: initialization, signing, and verification.

At the initialization phase, the scheme first establishes a Diffie-Hellman public-key system for both signer and verifier. At the signing phase, the signer runs the (Sign) algorithm using the message, its public key, and the Verifier public key as parameters to generate the signature. At the verification phase, the Verifier applies the (Verify) to recompute the secret based on its public key, the Signer public key, and the signature. If the secret and the message are the same, the authentication is accepted; otherwise it is rejected.

Initialization

1. Setup : choose g as primitive element in $GF(q)$ using security parameter k,
2. SKeyGen: choose Signer secret key sk_A and compute $pk_A = g^{sk_A}$,
3. VKeyGen: choose Verifier secret key sk_B and compute $pk_B = g^{sk_B}$

Sign (Signer A):

1. let
$$\alpha = pk_B^{sk_A}\,, \qquad \beta = pk_A$$
2. choose $t \in_R [1, q-1]$
3. compute the secret $s = H(m||\alpha^t)$
4. design the polynomial
$$f(x) = f_0 + f_1 x + f_2 x^2$$
such that
$$f(0) = s\,, \qquad f(1) = \alpha^t\,, \qquad f(2) = \beta$$
(There are three equations and three unknowns, so there is a unique $f(x)$.)
5. compute the signature $\sigma = f(3)$.
(the triplet: α^t, β and σ are shares of (3,3) Shamir scheme with the secret s)
6. transmit (m, σ, t) to the verifier.

Verify (Verifier B):

1. fetch pk_A
2. compute $\alpha = pk_A^{sk_B}$
3. compute the secret $s' = H(m||\alpha^t)$
4. compute the secret s based on the triplet α, β, t and σ
5. if $s = s'$ then accept, otherwise return $\perp$

9.4.4 Security Analysis

Theorem 9.1 *Let $\mathcal{A}$ be an* $\mathsf{EF}-\mathsf{CMA}-$*adversary against* DVS *in the random oracle model, which produces an existential forgery with probability* $\varepsilon = \boldsymbol{Succ}^{\mathsf{ef}-\mathsf{cma}}_{DVS,\mathcal{A}}(k)$*, within time* t*, making* $q_{\mathcal{H}}$*,* q_{Σ} *and* q_{Υ} *queries to the hash oracle* $\mathcal{H}$*, to the signing oracle* Σ *and to the verifying oracle (respectively). Then there exist* $\acute{\varepsilon} \in]0,1[$ *and* $\acute{t} \in N$ *verifying:*

$$\left(\begin{array}{l} \acute{\varepsilon} \geq \varepsilon - \frac{(q_H+q_\Sigma)q_\Sigma+q_\Upsilon}{2^k} \\ \acute{t} \leq t + (q_H+q_\Sigma)((q_H+q_\Sigma)T_{DDH} + T_{Exp} + q_\Upsilon(T_{poly}+T_{DDH})) + q_H T_{Poly} \end{array} \right)$$

such that GDH can be solved with probability $\acute{\varepsilon}$*, within time* $\acute{t}$*; where* T_{Poly}*,* T_{Exp} *and* T_{DDH} *denote the time complexity for constructing the polynomial* f*, time complexity for evaluating an exponentiation in* G*, and the time complexity of the DDH oracle, respectively.*

Proof. A proof of security is a polynomial time reduction from solving a well-established problem to breaking the **GDH** cryptographic primitive. The proof of security is carried out in the random oracle model [17]. In this model, hash functions are idealized as oracles which output a random value for each new input value.

The goal here is to prove that breaking the scheme is as difficult as breaking the Gap-Diffie-Hellman (GDH) problem. The idea of the proof is to simulate the overall outer environment of the adversary, and to take control of all the oracles to whom the adversary has access. We adopt the Shoup method [142] in using games to prove the security of the scheme. Each game is a variation of the previous one, beginning with the original real attack game, that is, the adversary faces the idealized oracles. In each of the following games, we replace the oracles by simulation with slight modifications so that it introduce some useful information to the adversary to help break the hard problem. We have to take care to make the underlining distributions indistinquishable from the real ones. Roughly speaking, the adversary does not see the difference in behaviour between the random and real games.

Since we apply a randomized signature generation, the security of the scheme is tightly related to the Gap-Diffie-Hellman assumption. However, the underlying construction is based on Shamir secret sharing; and the security of Shamir secret sharing has been discussed in [139].

Let (u, v) be a random instance of the *GDH* problem. We will simulate a machine which computes g^{uv} using Decisional Diffie-Hellman oracle, if we note that $U = g^u$ and $V = g^v$.

Game_0 We consider an EF-CMA-adversary $\mathcal{A}$ outputting an existential forgery with probability $\mathbf{Succ}^{\mathsf{ef-cma}}_{DVS,\mathcal{A}}$, with time t. The key generation algorithm (for the signer and for the verifier) is run once and it produces two pairs of keys (sk_A, pk_A) and (sk_B, pk_B). The adversary $\mathcal{A}$ is fed with pk_A and pk_B, and querying the random oracle $\mathcal{H}$, the signing oracle Σ, and the verifying oracle Υ, outputs a triple (m^*, σ^*, t^*), such that (m^*, σ^*, t^*) has not been obtained from the signing oracle.

We denote by $q_{\mathcal{H}}$, q_Σ and q_Υ the number of queries from the random oracle, signing oracle, and the verifying oracle respectively. For a signing query on a message m, we first ask a hash value of m and when the adversary outputs its forgery, one furthermore checks[2] whether it is actually valid or not. Therefore at most $q_{\mathcal{H}} + q_\Sigma$ are asked to the hash oracle and at most $q_\Upsilon + 1$ queries are asked to the verifying oracle during this game. In any Game_j, we denote by $\mathsf{Forge}_\mathsf{j}$ the event $\mathsf{DVS.Verify}(m^*, \sigma^*, pk_A, sk_B) = 1$. By definition we have:

$$Pr[\mathsf{Forge}_0] = \mathbf{Succ}^{\mathsf{ef-cma}}_{DVS,\mathcal{A}}$$

Game_1 We modify the simulation by replacing $\mathbf{pk_A}$ by U and $\mathbf{pk_B}$ by V. The distribution of (sk_A, pk_A, sk_B, pk_B) is unchanged since (U, V) is a random instance of the GDH problem. Note that the probability of both random variables U and V are the same if the public keys are chosen at random in the group G (which is obviously the case if G a q-prime order group generated by g, and the public keys are computed as g^x with x chosen at random in $[1, q]$. Therefore:

$$Pr[\mathsf{Forge}_1] = Pr[\mathsf{Forge}_0]$$

Game_2 In this game, we simulate the random oracle $\mathcal{H}$ and maintain an appropriate list, denoted $\mathsf{H\text{-}List}$. For any query $(m, \alpha) \in \{0,1\}^* \times \langle g \rangle$ we do the following:

- we check whether the random oracle $\mathsf{H\text{-}List}$ contains a quadruple $(m, \alpha, \perp, s)$. If it does, we output s as the answer to the oracle call.
- if not, we browse the signing oracle $\mathsf{H\text{-}List}$ and check for all 5-tuple $(m, \perp, s, \sigma, t)$ for whether α, U, V^t is a valid Diffie-Hellman triple. If it is, we give s as answer, and have a solution to our problem

[2] extra check to simplify the proof; otherwise, extra steps are needed to subtract from the success probability

- if not, we look for a $(m, \perp, \perp, \sigma, t)$, test whether α, U, V^t is a valid Diffie-Hellman triple, and if it is, we compute s $f(0)$ after having computed f such that $f(1) = \alpha, f(2) = V$, and $f(3) = \sigma$;
- otherwise, we pick at random $s \in \langle g \rangle$, record $(m, \alpha, \perp, s)$ in the H-List, and output s as the answer to the oracle call. In the random oracle model, this game is clearly identical to the previous one. Therefore, we get:

$$Pr[\mathsf{Forge}_2] = Pr[\mathsf{Forge}_1]$$

Game_3 In this game, we simulate the signing oracle Σ for any message m whose signature is queried to Σ by the adversary. We pick $t \in [1, q-1]$ at random.

- If the H-List of Σ includes a 5-tuple $(m, ?, ?, \sigma, t)$, we abort the simulation;
- else, we browse the H-List of $\mathcal{H}$ and check for each quadruple $(m, \alpha, \perp, ?)$, whether (α, U, V^t) is a valid Diffie-Hellman triple. If it does, we abort the simulation;
- otherwise, we pick $\sigma \in \langle g \rangle$ and add the 5-tuple $(m, \perp, \perp, \sigma, t)$ to the H-List and (σ, t) as the signature of m.

Since there are at most $q_{\mathcal{H}} + q_\Sigma$ messages queried to the random oracle $\mathcal{H}$, the new simulation aborts with a probability of at most $(q_{\mathcal{H}} + q_\Sigma)2^{-k}$. Otherwise, this new oracle perfectly simulates the signature. Summing up for all signing queries, we obtain

$$|Pr[\mathsf{Forge}_3] - Pr[\mathsf{Forge}_2]| \leq \frac{q_{\mathcal{H}} + q_\Sigma q_\Sigma}{2^k}$$

Game_4 In this game, we simulate the verifying oracle Υ. For any pair message/signature (m, σ, t) whose verification is queried

- we check whether the H-List includes a 5-tuple $(m, ?, s, ?, ?)$. If it does not, we reject the signature,
- for each $(m, \alpha, \perp, s)$ in the H-List we construct the polynomial $f(x)$ such that $f(2) = U, f(0) = s$, and $f(1) = \alpha$; we accept the signature if and only if there exists a f such that $f(3) = \sigma$ and α, U, V^t is a valid Diffie-Hellman triple;
- if the H-List includes 5-tuple $(m, \perp, \perp, \sigma, t)$, we accept the signature.

This simulation makes a difference only in the first step if σ is a valid signature on m, while (m, α) has not been queried from $\mathcal{H}$. Since $\mathcal{H}(m, \alpha)$ is uniformly distributed, the equality $s = \mathcal{H}(m, \alpha)$ happens with probability 2^{-k}. Summing up for all verification queries, we get

$$|Pr[\mathsf{Forge}_4] - Pr[\mathsf{Forge}_3]| \leq \frac{q_\Upsilon}{2^k}$$

The result output signature (m^*, σ^*, t^*) has not been obtained from Σ. From σ^*, V, s^*, we can compute f such that $f(1) = g^{uvt^*}$, and so $f(1)^{1/t^*}$ gives a solution to the GDH problem.

Summing up the above inequalities, we obtain:

$$\mathbf{Succ}^{\mathsf{gdh}}_{\mathsf{Game}_4}(k) \geq \mathbf{Succ}^{\mathsf{ef-cma}}_{DVS,\mathcal{A}}(k) - \frac{(q_\mathcal{H} + q_\Sigma)q_\Sigma + q_\Upsilon}{2^k}.$$

□

Note that the hashed message is the result of the concatenation of the message and the value α in order to associate the sender's tag with the message to prevent possible replay attacks. Only the verifier who established the Diffie-Hellman public-key with client is able to generate the value α. Obviously, the scheme does not provide non-repudiation service, but this is not a concern in our application for mutual authentication between a host and a trusted group manager.

9.4.5 Performance Issues

Concerning the running time, we have:

$$\acute{t} \leq t + (q_\mathcal{H} + q_\Sigma)((q_\Sigma + q_\mathcal{H})T_{DDH} + T_{Exp} + q_\Upsilon(T_{Poly} + T_{DDH})) + q_\mathcal{H}T_{Poly}$$

where T_{Poly}, T_{Exp} and T_{DDH} denotes the time complexity for constructing f, time complexity of the exponentiation, and the complexity of the DDH oracle respectively.

The operations involved at the signing and verification phase are almost the same:

- one cheap hash function on the message
- a number of flops (of multiplications and divisions of Lagrange interpolation function)
- one modular exponentiation

The modular exponentiation would not be expensive if we relax the security condition and set short key sizes without degradation. The overall signature size is 160 bits when using SHA-1 hashing.

9.5 Digital Signcryption

The scheme in the previous section provides peer authentication between peer entities. The proposed scheme in this section provides authenticity and privacy as well as integrity and identifiability all-in-one shot.

The first attempt to combine more than one security service in a single algorithm was by Zheng [169] at Crypto 1997 in his pioneer work. He addressed the question of the cost of secure and authenticated message delivery: whether it is possible to transfer a message of arbitrary length in a secure and authenticated way with an expense less than that required by signature-then-encryption. There, the goal was to provide simultaneously encryption and digital signature in a single step and at a cost less than individually signing and then encrypting. His motivation was based on an observation that signature generation and encryption consumes processor cycles, and also introduces "expanded" bits (i.e. size) to an original message. Hence, the cost of cryptographic operation on a message is typically measured in the message expansion rate and computational time invested by both sender and recipient. With typical standard sign-then-encrypt, the cost of delivering a message in a secure and authenticated way is essentially the sum of the cost of digital signature and that of encryption. The answer to the question in [169] was proposed by an approach based on the discrete logarithm problem of a shortened form of El-Gamal based signatures. In this approach, the secret key k was divided into two short sub-keys k_1 and k_2; the first was used for encryption, and the latter for signing. Zheng left as an open and challenging problem the design of other signcryption schemes employing any public-key cryptosystem such as RSA, or any other computationally hard problem. Later, in [152], Steinfeld and Zheng introduced another signcryption scheme based on the problem of integer factorization. The problem was also expanded to symmetric-key setting in other works [94]. Other studies by Bellare *et al* [16, 14], An *et al* [8], and others [123] have studied the fundamentals of this cryptographic primitive and have set its security proofs.

Our proposed scheme provides more services than typical signcryption. It provides not only privacy and authenticity, but also integrity. The transmitted signature requires minimum space overhead, hence it is superior in saving the bandwidth, especially in congested communication channels. The verification is efficient and requires relatively small computation time. The scheme is useful in applications which require strong security, yet with low space overhead. Our design is focused on authentication of the joining operation to a multicast group and on providing confidentiality as well.

9.5.1 Model of Security

We briefly review the security notions for encryption and signature schemes.

Signature Scheme

Description As described earlier, digital signature scheme $\sum$ is defined as

$$\sum = (\mathsf{GenSig}, \mathsf{Sig}, \mathsf{Ver}).$$

Security Notions Attacks against signature schemes can be classified according to the goals of the adversary and to the resources that it can use. The strongest (i.e. vital and difficult) attack is called an *existential forgery*, which means the attacker can provide a single message/signature pair. When the scheme prevents this kind of forgery, it is said to be *Non Existentially Forgeable* (NEF). When the attacker has access to a list of valid messages/signatures pairs, then the attack is called a *known-message attack* (KMA). However, if this list contains messages randomly and uniformly chosen, the attack is then termed a *random message attack* (RMA). Finally, the message may be chosen, adaptively, by the adversary himself, and this is called a *chosen-message attack* (CMA).

Public-Key Encryption

Description Asymmetric encryption or public-key encryption scheme $\prod$ is defined by three algorithms:

- GenEnc, the *key generation algorithm* which, on input 1^k, where k is the security parameter, produces a pair (pk,sk) of public and private keys;
- Enc, the *encryption algorithm* which, on input of a plaintext m and a public key pk, outputs a ciphertext c;
- Dec, the *decryption algorithm* which, on input of a ciphertext c and private key sk, outputs the associated plaintext m (or $\perp$, if c is an invalid ciphertext).

Security Notation A strong security notion is so-called *semantic security*, or what is also called *indistinguishability of encryptions* [14], IND. This means that if an attacker has some information about the plaintext, the view of the ciphertext should not leak any additional information. This security notion more formally considers the advantage an adversary can gain when trying to guess which of two messages which has been

encrypted. In other words, an adversary is seen as a 2-stage Turning machine (A_1, A_2), and the advantage $\mathsf{Adv}^{\mathsf{ind}}_{\prod}(\mathcal{A})$ should be negligible for any adversary, where

$$\mathsf{Adv}^{\mathsf{ind}}_{\prod}(\mathcal{A}) = 2 \times \Pr\left[\begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}), (m_0, m_1, c) \leftarrow A_1(\mathsf{pk}), \\ b \in \{0,1\}, c = \mathsf{Enc}_{\mathsf{pk}}(m_b) : A_2(m_0, m_1, s, c) = b\end{array}\right] - 1$$

On the other hand, an attacker can use many kinds of attacks, depending on the information available to him. First, in the public-key setting, the adversary can encrypt any message of its choice with the public key: this scenario is called *chosen-message attack*, and is denoted by CMA. An extended scenario allows the adversary restricted or unrestricted access to various oracles. The strongest attack is the *chosen-ciphertext attack* scenario denoted CCA2 [14] which can be accessed adaptively on the decryption oracle.

9.5.2 Formal Definition

Description: A signcryption scheme SC provides joint encryption and signing and is defined by three algorithms:

- Gen, the key generation algorithm which, for a security parameter k, outputs a pair of keys (SDK,VEK). SDK is the user's sign/decrypt key, which is kept secret, and VEK is the user's verify/encrypt key, which is made public.
- SigEnc, the encryption and signing algorithm which, for a message m, the public key of the receiver VEK_B, and the private key of the sender SDK_A, produces $\sigma = \mathsf{SigEnc}_{\mathsf{SDK},\mathsf{VEK}}(m)$
- VerDec, the decryption and verifying algorithm which, for $s-$*ciphertext* σ, the private key SDK_B of the receiver, and the public key VEK_A of the sender, recovers the message $m = \mathsf{VERDEC}_{\mathsf{VEK},\mathsf{SDK}}(\sigma)$. If this algorithm fails to recover the message or to verify authenticity, it returns $\perp$.

Security Notions For the security notions of a signcryption, one can combine the classical ones from signature [66] and from encryption [14] under adaptive attacks. With an access to the public information, $\mathsf{PUB} = (\mathsf{VEK}_A, \mathsf{VEK}_B)$, and oracle access to the functionalities of both A and B (i.e. access to the *signcryption* and to the *de-signcryption* oracles), the adversary should be able to break:

- authenticity NEF – produce a valid *s-ciphertext* of a new message, and thus provide an *existential forgery*;

- confidentiality IND - break the *indistinguishability* of *c-ciphertext*

Definition 9.5 *A signcryption scheme is secure if it achieves IND/NEF under adaptive attacks.*

For any adversary $\mathcal{A}$ that runs an adaptive attack, we denote by $\mathsf{Adv}_{\mathsf{SC}}^{\mathsf{ef-cma}}(\mathcal{A})$ its probability of success in forging a new valid *s-ciphertext*. In the same way, we denote by $\mathsf{Adv}_{\mathsf{SC}}^{\mathsf{ind-cca}}(\mathcal{A})$ its advantage in distinguishing *c-ciphertexts*.

9.5.3 Description

The main objective of signcryption is to provide both encryption and authentication. In the SC scheme, for encryption, an efficient public-key cryptosystem algorithm $\prod$ is used, and for authentication, a signing scheme $\sum$ is used. The secret sharing method is used as a template to include both objectives in terms of shares. As described in section 9.3, the idea is to fix a secret, and to relate to the secret a number of parameters or shares. To achieve this, we need to define parameters (shares) for the system. Here, we use the (2,2) threshold secret sharing method. The sender A constructs the polynomial and secret. The challenge for the receiver B, therefore, is to reconstruct the secret based on the knowledge of the received values.

The building blocks of the SC signcryption scheme comprises the following primitive cryptographic components:

- an encryption public-key cryptosystem $\prod = (\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$,
- a digital signature scheme $\sum = (\mathsf{GenSig}, \mathsf{Sig}, \mathsf{Ver})$,
- a one-way hash function H,
- threshold Shamir secret sharing SSS.

9.5.4 The SC Scheme

$\underline{\mathsf{Gen}(1^k) = \mathsf{GenSig}(1^k) \times \mathsf{GenEnc}(1^k)}$

One first gets $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{GenSig}(1^k)$ and $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{GenEnc}(1^k)$.
Then, let $\mathsf{SDK} = (\mathsf{sk}_1, \mathsf{sk}_2)$ and $\mathsf{VEK} = (\mathsf{pk}_1, \mathsf{pk}_2)$.

$\underline{\mathsf{SigEnc}_{\mathsf{SDK}_A,\mathsf{VEK}_B}(m) : A}$

1. compute $h = \mathsf{H}(m)$
2. compute $s = \mathsf{Sig}_{\mathsf{SDK}_A}(h)$
3. compute $c = \mathsf{Enc}_{\mathsf{VEK}_B}(m)$
4. design the polynomial
$$f(x) = f_0 + f_1 x$$
such that
$$f(0) = c\,, \qquad f(1) = s$$
(There are two equations and two unknowns so there is a unique $f(x)$.)
5. compute the signature $\sigma = f(2)$.
(the couple: σ and s are shares of (2,2) SSS with the secret c)
6. transmit (s, σ) to the receiver.

$\underline{\mathsf{VerDec}_{\mathsf{VEK}_A,\mathsf{SDK}_B}(s, \sigma) : B}$

1. compute the secret c based on the values s and σ
2. compute $m = \mathsf{Dec}_{\mathsf{SDK}_B}(c)$
3. compute $h = \mathsf{H}(m)$
4. compute $h' = \mathsf{Ver}_{\mathsf{VEK}_A}(s)$
5. if $h = h'$, then accept, otherwise return $\perp$

9.5.5 Security Issues

The sender constructs the signature σ in an elegant way using the secret sharing technique. The original message and its ciphertext are compact in σ. The receiver reconstructs the value c from shares s and σ by solving a set of equations using the Lagrange interpolation function. Once c is retrieved, m could be retrieved as well by decrypting the value c. Consequently, it would be possible to compute the hash of the message m to get the value h. The decryption of the value s would result in the value h'. If the computed hash value h from m is the same as the value h' resulting from decrypting the value s, then this proves the authenticity of the message sent through the pair (s, σ).

The transmitted message is not clear for opponents and can be revealed only by the designated receiver. The adversary, in the worst case, can only reveal the ciphertext c. Therefore, the known attacks to manipulate clear messages to find collisions to signatures are not possible. If we assume that an adversary was able to manipulate either the cipher message s or the signature σ itself, then it would be easily found by the integrity check of the derived message digest h. The verification which is performed on the signature σ is achieved by the secret sharing technique, which adds extra security privilege to the system. The identity of the signer could be derived since the signer is signing by its private key.

Hence, the scheme provides all-in-one security services. It provides confidentiality, authenticity and integrity. Yet, under certain conditions shown, it is efficient compared to the services and to other schemes. For efficiency purposes, we relax the security condition and use short keys, and the security objectives are still retained.

The security of the overall signcryption scheme depends on the security of the underlying building block components, and specifically:

1. The public-key cryptosystem is used to encrypt the message m to c. The choice of selecting a specific public-key cryptosystem $\prod$ was not declared. This property gives the system more design flexibility. However, we assume $\prod$ to be semantically secure against a chosen cipher attack, i.e. $\mathsf{IND}-\mathsf{CCA2}$. For the sake of efficiency, and also considering the security, we may relax some security properties of the underlying structures. In particular, we relax the security properties of the asymmetric key system and choose short keys without security deterioration.

   Proposition 9.2 *Let* $\prod$ *be a* $\mathsf{IND}-\mathsf{CCA2}$ *secure asymmetric encryption scheme; then any scheme* $\mathsf{SC}_{\Sigma,\Pi}$ *associated with* $\prod$ *is* $\mathsf{IND}-\mathsf{CCA2}$ *secure.*

More precisely, for any $\mathsf{IND}-\mathsf{CCA2}$ adversary $\mathcal{A}$ which takes advantage $\mathsf{Adv}^{\mathsf{ind}-\mathsf{cca}}_{SC_{\Sigma,\Pi,\mathcal{A}}}$ against $\mathsf{SC}_{\Sigma,\Pi}$ within time t, making $q_{\mathcal{H}}, q_{\Sigma}$, and $q_{\mathcal{D}}$ queries to the random oracles, the signing oracle, and the decrypting oracle respectively, there exists an $\mathsf{IND}-\mathsf{CCA2}$ adversary $\mathcal{A}'$ against $\prod$, making $q_{\mathcal{H}}$ queries to the random oracles, $q_{\mathcal{D}}$ queries to the decrypting oracles, within time $t + O(1)$, which has the same advantage as $\mathcal{A}$:

$$\mathsf{Adv}^{\mathsf{ind}-\mathsf{cca}}_{\Pi,\mathcal{A}'} = \mathsf{Adv}^{\mathsf{ind}-\mathsf{cca}}_{\mathsf{SC}_{\Sigma,\Pi,\mathcal{A}}}$$

This property ensures the semantic security of SC scheme.

2. The digital signature scheme is used to sign the message. Again, the choice of selecting the scheme was not specified. However, we assume a digital signature scheme which is secure against existential forgery of chosen message attack. As a result, we achieve the following:

Proposition 9.3 *Let Σ be an $\mathsf{EF}-\mathsf{CMA}$-secure digital signature scheme, then any scheme $\mathsf{SC}_{\Sigma,\Pi}$ associated to Σ is $\mathsf{EF}-\mathsf{CMA}$ secure.*

More precisely, for any $\mathsf{EF}-\mathsf{CMA}$, adversary $\mathcal{A}$ which takes advantage $\mathbf{Adv}^{\mathsf{ef}-\mathsf{cma}}_{\mathsf{SC}_{\Sigma,\Pi,\mathcal{A}}}$ against $\mathsf{SC}_{\Sigma,\Pi}$ within time t, making $q_{\mathcal{H}}, q_{\Sigma}$, and $q_{\mathcal{D}}$ queries to the random oracles, the signing oracle, and the decrypting oracle respectively, there exists an $\mathsf{EF}-\mathsf{CMA}$ adversary $\mathcal{A}'$ against Σ, making $q_{\mathcal{H}}$ queries to the random oracles, q_{Σ} queries to the signing oracles, within time $t + O(1)$, which has the same advantage as $\mathcal{A}$:

$$\mathbf{Adv}^{\mathsf{ef}-\mathsf{cma}}_{\Sigma,\mathcal{A}'} = \mathbf{Adv}^{\mathsf{ef}-\mathsf{cma}}_{\mathsf{SC}_{\Sigma,\Pi,\mathcal{A}}}$$

This property ensures the existential unforgeability of the SC scheme against chosen message attack.

3. The security of the hash function. We strongly emphasize the selection of a collision-resistant one-way hash function H which takes an arbitrary size string and generates a string of size l: $H : \{0,1\}^* \rightarrow \{0,1\}^l$. SHA-1 is proven to be a secure message digest algorithm. It generates 160 bits of message digest, which makes it secure enough against well-known attacks.

4. The security of Shamir secret sharing SSS. The security of Shamir secret sharing is discussed in [139]. There, the assumptions and requirements for security of the

system are listed. The main concern of the security is related to the perfectness of the secret sharing method. In general, the system is perfect when shares are selected randomly. In the scheme, the system is of degree one with two parameters. Thus, at least one random share is needed. The other share could be selective without any security side effect. The chosen share in our scheme is the secret c. However, for the random value, we consider s is the random share because, from a security perspective, the system would be as secure with s as it is secure with a real random share. This claim is true since the share s is actually a signed parameter by a private key. The share σ is computed from the other shares, and hence it is secure.

9.5.6 Performance Issues

The operations involved at the signing and verification phase are almost the same:

- encryption/decryption operation. We assume a very efficient public-key algorithm. The type of algorithm is non-important.
- one cheap hash function on the message
- a number of flops (of multiplications and divisions of the Lagrange interpolation function)

Generating and extracting the secret c from the system requires a number of operations including multiplications and divisions. Generally, the overall Lagrange operations are of $O(n^2)$. Specifically, they include $(n+1)$ multiplications for n times. Also, a number of $(n+1)$ divisions are required. In our system, we have 3 parameters; thus, the approximate number of operations is almost 15 flops per message. Also, there is one hash computation of the message. The modular exponentiation would not be expensive if we relax the security condition and set short key sizes. The performance also depends on with the size of the message m and the key to compute the value c. We already assumed short keys without losing security robustness. So the signing and verification process should be fast. The overall signature size is 160 bits according to SHA-1 message digest and 160 bits for the value σ. Since more security services are included, the total cost is higher than the cost of signcryption schemes.

Note that the values s, h and c are in the same ring Z_p and hence have the same size. In fact, this equals the size of the digest resulting from the hash function; for

example, the SHA-1 hash function outputs 160 bits of message digest. This size is sufficient for our multicast joining case as the purpose is to authenticate with privacy a specific and short message of the joining host such as its group-name or user-name. A general purpose scheme, using the same idea above but with a different technique, was developed by Pieprzyk and Pointcheval in [123].

9.6 Summary

In this chapter, we have exploited the idea of the Shamir threshold secret sharing method to develop two variants of digital signatures. The first scheme proposes a designated verifier signature to allow a specific verifier to verify the signature of a specific signer. In the context of group communication, this scheme could be used as a cryptographic protocol for peer authentication to allow a verifier to verify the authenticity of the host before joining a group.

In the second scheme, we used the same idea to develop a protocol to provide authenticity, privacy, and integrity all-in-one shot and with less computational cost as well as communication overhead. The scheme is flexible and is not restricted to any particular cryptosystem, and it could be designed with any cryptographic cryptosystem as far as it satisfies security requirement. This primitive is known as "signcryption". In the context of group communication, the protocol could be used in cases where the joining operation requires not only authenticity, but also privacy and integrity.

Chapter 10

Conclusion and Future Directions

Source authentication is a major security service in modern communication networks. For broadcasting applications such as pay-TV it is an essential requirement for receivers to receive authentic information. It is always possible for an adversary or enemy to either inject, intercept, tamper with, or reproduce the original stream of broadcasting. This is a crucial issue in the contemporary communication services, especially with the current advances in telecommunication technology on one side, and the new techniques used by hackers in breaching communication systems on the other.

Given this concern, modern communication systems should be provided with a sophisticated security infrastructure that is designed to provide complete information protection. To achieve this requirement in a multicast environment, clients (or users) of a service must first register themselves with the service provider. Based on successful registration, service providers can provide authentic and private information. Not only this, but clients may also need to send data back to the original senders, who become in this case the receivers. They in turn need their information to be secure.

The goal of this thesis, when it was started in the beginning of year 2000 and during the early days of the prolific research in multicast security, was to build a complete authentication system. By 'complete', we mean that all communications involved in the group environment should be authenticated. We always believed that partial secured systems would not provide enough security to systems. Therefore, our theme in this thesis was "building an ideal authentication system".

The methodology we followed was studying multicast technology and the security of multicast as well as the related communication paradigms. We focused in particular on source authentication. Authentication in group and multicast communication is a challenging problem as one needs to consider the efficiency and security which is either not practical or not provided in the conventional authentication methods. Our methodology was to study the problem from all directions. We also looked into the defacto modes of communications to multicast. We first reviewed the literature and

state-of-the-art techniques. Then we filled the gap and considered unexplored situations and scenarios. Our strategy was to propose, whenever possible, more than a solution to one single case problem.

The contributions in this thesis consists of more than twelve cryptographic protocols along with security and efficiency analysis or discussion. The contents of this thesis were published as academic papers in prestigious conference proceedings and appeared in the *Lecture Notes of Computer Science* series as well as in *Kluwer Academic Publisher*. Typically, the papers were reviewed and refereed by at least three international experts in cryptography and networks. The cryptographic protocols were designed to solve difficult group authentication problems. Some of our papers, contributing this thesis, were the key ideas and motivations for other author papers, for example the concast signature, the proxy one-time signature, and the group threshold one-time signature. The other topics such as authentication of anycast communication was the first academic paper to discuss this issue. We started the topic of authenticating of transit flows as further open for investigation as no such protocol existed to our best knowledge at that time. Our work in designated verifier signature and signcryption coincides with other works in the same topic but with different approaches, scope and purposes. We are the first to use the secret sharing method rather than the discrete logarithmic problem.

We covered conventional digital signatures, group signatures, multi-signatures, proxy signatures, one-time signatures, batch signatures, designated-verifier-signature and others. Also, we invented a new sort of digital signatures such as concast signatures, k-siblings one-time signatures, secret-sharing-based signatures such as the signcryption and the designated verifier signature. The contributions extended also to solving open problems such as the one in Section 5.5. For the first time, One-time signatures were used for proxy and threshold society-oriented cryptography.

10.1 Summary

The theme of this thesis is developing cryptographic protocols for authentication group communications. The thesis starts with an introduction of the problem under study. In Chapter 2, we give an overview of cryptographic primitives used for building authentication protocols such as the public key cryptosystem, private key cryptosystem, hashing, digital signature, and secret sharing schemes.

Then, in Chapter 3, an overview of multicast communication networks was given,

along with discussion of its security related issues. Chapters 4, 5, 6, 7 , 8 and 9 were dedicated to the contributions.

In Chapter 4, we have introduced authentication schemes based on linear equations and used combinatorial designs to authenticate multicast streams. In Chapter 5, we proposed three different authentication schemes for the concast communication mode. Also, we suggested a solution to the open problem of finding a fast screening signature for non-RSA digital signature schemes. In Chapter 6, we exploited the siblings hashing method to design a new authentication scheme that can verify the authenticity of transit messages at the intermediate nodes in a communication network. We have also used it to design a new one-time digital signature which has low computation and space overhead. In Chapter 7, we develop techniques to use one-time signatures in threshold and proxy group communication. In Chapter 8, we studied the authentication of a special sort of group communication know as anycast communication, as the de facto to multicast communication. In Chapter 9, we proposed a new designated verifier signature and new signcryption schemes based on the Shamir threshold secret sharing technique. Both were used in the context of authenticating peer communication.

10.2 Future Directions

There are several topics that are the subject for ongoing investigation by researchers in the area of source authentication in group and multicast communication. We indicated some of them at the summary section of the chapters and we outline them and other directions in this section.

Batch Signatures for Group Authentication

The extension of the threshold scheme in Chapter 4 may be developed into a fast batch verification algorithm for a digital signature algorithm that considers the order of received packets [77, 76, 102, 166].

Authentication of Many-to-Many Communication

Another interesting problem related to group communication is many-to-many communication. Multiple source groups have special requirements for denial-of-service attack protection and for minimizing states needed for sender authentication. There is only one paper so far on this problem [55].

Authentication Using Secret Sharing Methods

In Chapter 9, we considered Shamir secret sharing techniques. A possible authentication technique may be investigated for other secret sharing techniques such as the Blakley scheme [22]. Many versions of verifiable secret sharing techniques may be further exploited for authentication purposes [150].

Authentication of Transit Flows

We shed light on this unexplored problem. We proposed the authentication of transit flows by hierarchical sibling and leave it as an open problem to evaluate and analyze its security and efficiency, or develop a more efficient protocol than the proposed are.

Signcryption for Concast Communication

In Chapter 5, we only discussed how to authenticate concast communication. An interesting problem would be to provide both authenticity and privacy, which is known as signcryption, to concast communication.