Chapter 1

Introduction

There has been great interest in the Internet society in the past few years in the area of group communication. This interest stems from the emergence of new sorts of applications that are based on group interaction or collaborative work among multiple users distributed over various locations. There has also been a remarkable increase in real-time streaming applications which have special quality of service requirements.

Examples of such emerging distributed, real-time and group-based applications are online video conferences, video/audio streaming, shared white-board, multimedia teleconferencing, remote consultation/diagnosis systems for medical application, e-learning, pay-TV, push media, and others.

These distributed and collaborative applications require scalable and efficient information exchange among the group members, and have fairly low latency message delivery of both small and large messages.

The traditional mode of communication used over any network transportation is unicast communication, which establishes a point-to-point link between two peers. The unicast communication mode for applications based on group interactions would be neither a scalable nor an efficient option. For example, for a group communication consisting of n users, $n \times n$ unicast communication links would need to be established. Therefore, there would be a considerable waste in network bandwidth and resources.

Because unicast failed to provide an efficient communication tool for group communications, there was a great need to establish a sophisticated infrastructure for group communication over a wide area network such as the Internet that would be capable of transporting data in a low bandwidth, with high speed, using an efficient mechanism, and in a scalable manner.

1.1 Multicast

In his pioneer work, Deering [47] established a framework for group communication in distributed systems. More precisely, he founded and developed a group communication architecture, known as *multicast*, in which a source can send data to a group of recipients in an efficient way.

Multicasting is a significant tool in a communications network because it enables applications to scale, thereby offering service to many users without overloading the network and, at the same time, preserving resources. Multicast became an important mode of communication as well as a good platform for building group-oriented service.

The main problem in building a multicast architecture was how to deliver a message to a group of recipients on a large-scale, dispersed network such as the Internet. In network terminology, how would it be possible to build an efficient routing protocol which could manage and deliver messages to distributed hosts in an efficient, scalable and reliable way? The mechanism used by multicast is to send one copy of the message (by the host); the multicast router would then duplicate the message to the group of recipients using multicast routing protocol.

Multicast routing requires the building of a distribution tree in which multicast data can be transmitted to the group members at the leaf of the distribution tree. The multicast distribution tree is shaped using a multicast routing tree such as CBT, DVMRP, MOSPF, PIM-SM, or PIM-DM. Any host can join a multicast group by using group membership protocol such as IGMP which directs their subnet router to join a multicast distribution tree. This allows IP multicast to scale to a large number of participant hosts.

Multicast is not usually a one-direction operation or sender-to-receivers mode. Often, receivers send their acknowledgments back to the sender. This reverse direction communication of multicast is called *concast*. Concast is a many-to-one communication service, sometimes referred to as a Report-in style application, and it usually takes the shape of receivers-to-sender.

There are also other communication modes which are multicast-related such as *anycast*. Anycast communication allows a source to transmit a message to a single destination node, out of a group of destination nodes. It is a sort of communication model related to redirecting the client to the "best" server among multiple content servers. The best is measured as the best among a combination of network criteria.

1.2 Multicast Security

In multicast communication, potential security threats are similar to those encountered in unicast transmissions. However, because of the inherent broad scope of a multicast session, the potential for attacks may be greater than for unicast traffic. The nature of multicast presents new problems that cannot be effectively dealt with using trivial extensions of techniques for secure unicast. For example, in setting up a secure point-to-point communication channel, one knows the identity of the part at the other end. In a multicast session, it is typically not guaranteed to know who are present at a session at a time.

Another side of the problem is group-oriented secure data exchange. Although using $O(n^2)$ end-to-end secure channels can provide secure group communication between n peers, such architecture loses all the advantage a multicast facility has over unicast.

Furthermore, the number of communication links traversed by wide-area network multicast can potentially be greater than compared with a single unicast, where the communication path is a collection of links and nodes.

Also, in group-oriented communication, an additional mechanism is needed to reliably establish the identity of the originator of the message. Moreover, group-oriented authentication and key distribution is not necessarily $O(n^2)$ pairwise authentication because of the many possible interpretations of the meaning of belonging to a multicast session. Therefore, new protocols are needed to perform security functions to control session organization and management, secure broadcast, and user access to the network.

What is needed actually is a trusted multicast architecture to handle these security issues in a coherent manner and to protect network services and applications. Apart from satisfying the security requirement mentioned earlier, a desirable design should also be compatible with existing network protocols, be scalable to global Internet, and be transparent to the other network layers. The architecture also has to be flexible to support a variety of policies.

In order to counter common threats to multicast communication, we can apply several of the fundamental security services, including authentication, integrity, and confidentiality. A secure multicast session may use all or a combination of these services to achieve the desired security level. One of the main security services is authentication.

Authentication services provide assurance of participant host identity. An authentication mechanism can be applied to several aspects of multicast communications.

Above all, authentication is an essential part in providing access control in keying material. Also, to identify the source of multicast traffic, an authentication mechanism can be applied by the traffic source. Authentication may serve further to establish group membership by identifying group members along with their data which are destined for the group members. Protocols such as IP Authentication Header (AH) [88] can provide authentication for IP datagrams, and they may be used for host authentication. Authentication is also an essential part of any key distribution protocol. Because of the sensitive nature of the keying material, the authentication mechanism can identify the source of the key material and provide a means to counter various masquerade and replay attacks that may be launched against a secure multicast session. Applying an authentication mechanism to transmitted multicast group data can also provide a strong level of integrity protection. These mechanisms provide not only a level of assurance to receivers on data origination, but they may also provide indication of data corruption.

1.3 Motivation and Methodology

A typical and complete multicast scenario starts when a host tries to join a multicast environment using a membership protocol and becomes a member of a group. Based on a successful membership, the host is able to send or receive messages to or from other members of the group. When a sender sends a message to a group of recipients, the operation takes the form of one-to-many communication. Consequently, receivers would send their acknowledgments back to the sender in many-to-one mode. Traffic may possibly pass through intermediate nodes in the network, between the sender and receiver as transit flows.

Since, in multicast, a group of participants are involved in the communication, source authentication is a major security service as a receiver of a message would not be able to know whether the message is sent from a legitimate sender in the group.

Our goal in this work is to provide a sound solution to the problem of authentication in multicast communication. We aim to provide an *ideal authenticated multicast environment*. Existing protocols for multicast offer only partial solutions. Our methodology is to study all the communication operations involved in multicast communication and propose cryptographic solutions for the authentication problem for all the communication operations. This is unlike other studies in this field, which have only focused on the sender-receivers side.

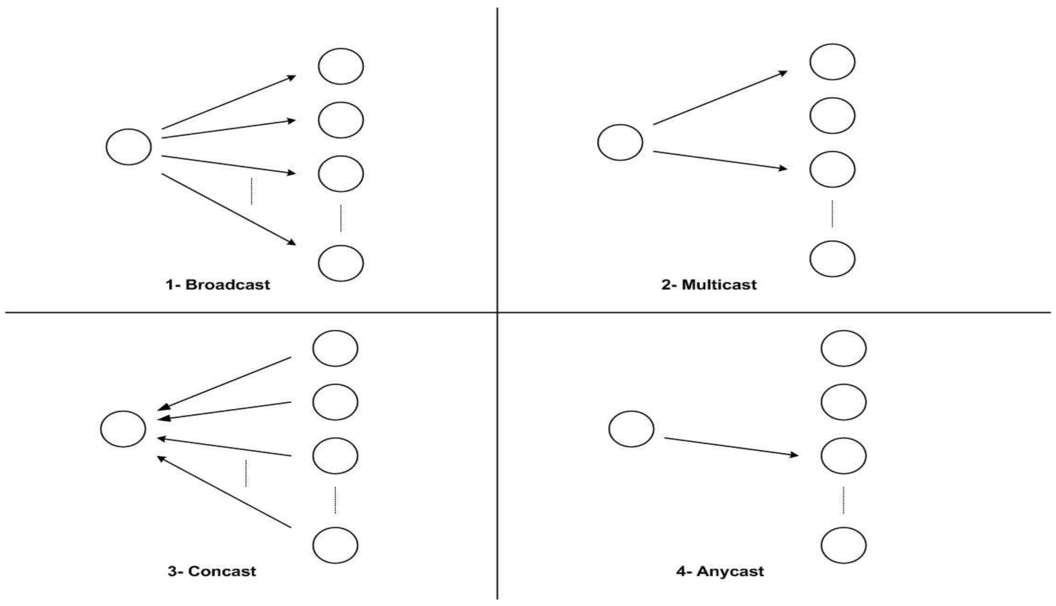

Figure 1.1: Group-based communication modes

We have divided the authentication process in a multicast environment into the following stages:

1. one-to-one (joining)
2. one-to-many (broadcasting)
3. many-to-one (concasting)
4. intermediate (transit)

Our approach is to find a new, innovative and efficient authentication scheme(s) at each stage. Each stage has a different communication mode; therefore, designing an authentication scheme should take this issue into consideration.

At the beginning, a host tries to join a multicast group. This is a very important stage, where the group manager should not allow a non-trusted host to join the group. Joining may be based on a cryptosystem such as public or private-key between host and multicast manager.

The authentication scheme at the one-to-many mode has to consider the non-repudiation service since group members share the same group key, and any member may re-send the message and masquerade as the original sender. Therefore, symmetric key cryptosystems are not the proper solution. On the other hand, typical digital signatures are costly operations.

At the many-to-one stage, the verification process should be efficient enough to handle the overwhelming number of messages that need to be processed simultaneously.

At the intermediate stage, verification is performed over different and independent datagrams.

In addition, we may also need to authenticate group-related communication modes such as those related to redirecting the client to multiple content servers, which is also known as anycast.

One of the techniques to improve the efficiency of authentication in group communication is to use one-time signatures, which can be used regardless of communication mode described above. This method emerged from the observation that typical authentication schemes such as digital signatures RSA [132] and ElGamal [57], have both high computational and high space overhead; hence, they do not fulfill the new requirements for most new applications. On the other hand, one-time signatures are more efficient than the typical one, and many applications, or new devices with limited power, are emerging that require efficient authentication evaluation as a result. Therefore, new schemes using one-time signatures for group-based applications are proposed which cannot only guarantee secure communication, but also maintain the efficiency of operations.

1.4 Organization of the Thesis and Contributions

To fully cover all aspects of the topic in order to make it self-contained, we present in the next two chapters a broad overview of the two joint fields of the study: cryptography and multicast networking.

Chapter 2 provides the necessary background in cryptographic essentials. This includes the introduction of private and public key cryptosystems, hashing, digital signature, and secret sharing scheme.

In Chapter 3, we give a necessary background of multicast technology, and specifically the motivation, evolution, structure, protocols and requirements. We also focus on multicast security, and in particular on key-management as it plays a vital role in information distribution. In conjunction with the remaining chapters, multicast security, with focus on authentication, is described.

In Chapter 4, we survey and classify the proposals of the stream authentication problem in the multicast environment and categorize them into a category which uses digital signatures, and another category which uses MAC. New approaches for authenticating digital streams using threshold techniques are introduced. The main advantages of the new approaches are toleration of packet loss, up to a threshold number, and having a minimum space overhead. These are most suitable for multicast applications running over lossy, unreliable communication channels while retaining the security requirement. We use linear polynomial and combinatorial design methods. The contents of this chapter is to be found in our paper [6], which was published in the ICN'01 (International Conference in Networking 2001)

In Chapter 5, we turn to the inverse operation of multicast communication: concast communication, the many-to-one communication mode. In this chapter, we propose several schemes for authenticating concast communication based on the trustability of the coordinator of the group. We use variants of sibling intractable hashing for trusted nodes, the ElGamal signature scheme for untrusted nodes, and batch signatures for networks of non-trusted modes. The contents of this chapter can be found in our paper [5], which appeared in the proceedings of INDOCRYPT'02, (Third International Conference on Cryptology in India 2002).

In Chapter 6, we study the possibility of authenticating messages passing through intermediate nodes between source and destination. 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. The contents of this chapter appeared in our paper [7], which has been published in the proceedings of the "Sixth International Conference on Communication and Multimedia Security 2002 " (CMS'02).

Chapter 7 is a continuation of section 6.5 on the topic of using a one-time signature. In this chapter, we focus on one-time signatures as an efficient cryptographic primitive for authenticating group or broadcast communication. That is, we attempt to apply one-time signatures for a situation where the right to execute signature operations is shared by a group of signers, with or without the aid of a trusted party. We further consider situations where such signing is accomplished with the aid of a proxy. To our best knowledge, the problem of finding a group-oriented one-time signature, as well as the problem of finding a proxy one-time signature, has not been discussed elsewhere. The contents of this chapter is to be found in our paper [4], which appeared in the proceedings of the "First MiAn International Conference in Applied Cryptography and Network Security 2003"(ACNS'03).

In Chapter 8, we focus on authentication of anycast communication as a multicast-related communication mode. This deals mainly with the problem of redirecting a client to multiple content servers. The contents of this chapter is to be found in our paper [3], which was published in the proceedings of MMM-ACNS-2003: the second international workshop in "Mathematical Methods, Models and Architectures for Computer Networks Security 2003".

In Chapter 9 we propose a secure one-to-one authentication scheme based on a secret sharing technique. The preliminary version of this protocol was published in our paper [1]. The development of this protocol to a signcryption scheme appeared in the proceedings of MMM-ACNS-2003: the second international workshop in "Mathematical Methods, Models and Architectures for Computer Networks Security 2003"[2].

Chapter 10 summarizes the work, and states directions of future work.

The chapters are organized according to the flow of information and correlation of topics.

Chapter 2

Cryptographic Essentials

Cryptography is the science that deals with the design of algorithms, protocols and systems for solving or providing particular kinds of security services such as *confidentiality*, *integrity*, *authentication* and *non-repudiation*.

- *confidentiality* is a service to keep the content of information secret from all but those authorized to see it.
- *integrity* is a service which addresses the unauthorized alteration of data.
- *authentication* is a service related to identification. Two or more parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to the origin and the data contents. Therefore, two types of authentication are known in Cryptology: *entity* and *source authentication*, respectively.
- *non-repudiation* is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, the non-repudiation service provides a means to resolve the dispute.

More precisely, cryptography is the study of mathematical techniques related to aspects of information security. Cryptography is also about the prevention and detection of cheating and other malicious activities. It also provides a designer with tools to implement the required information protection or authentication. *Applied cryptography* is about using the cryptographic primitives in developing security applications and protocols. The important parameter in almost all modern cryptographic systems is the *key*, which selects the particular transformation to be employed. The size of the key space should be large enough to preclude exhaustive search.

In this chapter, we present the essential cryptographic primitives necessary for designing the cryptographic protocols of the subsequent chapters.

2.1 Terminology

Cryptography has developed an extensive vocabulary. There is a collection of basic terms that are discussed briefly in this section. These definitions are taken from Menezes *et al.* [107]. Other terms will be introduced in this thesis as necessity arises.

A *party* is someone or something which sends, receives, or manipulates information.

The *sender* is the party in a communication system that is the legitimate transmitter of information.

The *receiver* is the party in a communication system that is the intended recipient of information.

An *adversary* is a party in a communication system that is neither the sender nor receiver, and which tries to defeat the information security service being provided between the sender and receiver.

Secrecy ensures that information flow between the sender and the receiver is unintelligible to outsiders. It protects information against threats based on eavesdropping.

Integrity enables the receiver to verify whether the message has been tampered with by outsiders whilst in transit via an unsecured channel. It ensures that any modification of the stream of messages will be detected.

An *identification* or *party authentication* assures the parties of their identity. Message authentication provides evidence of the identity of the sender to the party which receives a message.

A *channel* is a means of conveying information from one party to another.

A *secure channel* (or a private channel) is one from which an adversary does not have the ability to reorder, delete, insert, or read. An insecure channel (or a public channel) is one from which parties other than those for which the information is intended can reorder, delete, insert, or read.

Encryption is the primitive cryptographic operation used to ensure secrecy or confidentiality of information transmitted across an unsecured communication channel. The encryption operation takes a piece of information, also called message or plaintext, and transform it into a cryptogram or ciphertext using a secret cryptographic key.

Decryption is the reverse operation to encryption. The receiver who holds the correct secret key can recover the message (plaintext) from the cryptogram (ciphertext).

The *encryption algorithm* (or decryption algorithm) is the procedure that gives a step-by-step description of the encryption (or decryption) process. If there is no need to distinguish encryption from decryption, we call them collectively ciphers or cryptosystems.

Private-key (also called symmetric) cryptosystems use the same secret key for encryption and decryption. Although the encryption and decryption keys do not need to be identical, the knowledge of one of them suffices to obtain the other.

Public-key (also called asymmetric) cryptosystems use a different key for encryption and decryption. The knowledge of one key, however, does not allow the other to be determined.

A *one-way function* is a function for which it is easy to compute its value from its arguments(s), but it is "difficult" to reverse it, that is, to find its arguments(s) although knowing its value.

Cryptanalysis is the study of mathematical techniques for attempting to defeat information security services. A cryptanalyst is someone who engages in cryptanalysis.

Cryptology is the study of cryptography and cryptanalysis.

A *cryptosystem* or a *cryptographic system* is a general term referring to a set of cryptographic primitives used to provide information security services. Most often this term is used in conjunction with primitives providing confidentiality, that is encryption.

An encryption system is said to be *breakable* if a third party, without prior knowledge of the key, can systematically recover plaintext from corresponding ciphertext within some appropriate time frame. The objective of the following attacks is to systematically recover plaintext from ciphertext, or, even more drastically, to deduce the decryption key.

- *A ciphertext-only attack* is one where the adversary (or cryptanalyst) tries to deduce the decryption key or plaintext by observing ciphertext only. Any encryption scheme vulnerable to this type of attack is considered to be completely insecure.

- A *known-plaintext attack* is one where the adversary has samples of both the plaintext and its corresponding ciphertext, and is at liberty to make use of them to reveal further the secret key.

- A *chosen-plaintext attack* is one where the adversary chooses plaintext, and is then given corresponding ciphertext. Subsequently, the adversary uses the information deduced in order to recover plaintext corresponding to previously unseen ciphertext, or to find the key applied.

- *An adaptive chosen-plaintext attack* is a chosen-plaintext attack in which the adversary makes a series of interactive queries, choosing subsequent plaintexts based on the information from the previous encryptions.

- A *chosen-ciphertext attack* is one where the adversary selects the ciphertext, and is then given the corresponding plaintext. The objective is then to be able to deduce the plaintext from different ciphertext (not seen before).

- An *adaptive chosen-ciphertext attack* is one where the adversary sends a large number of ciphertexts to be decrypted, using the results of these decryptions to select subsequent ciphertexts, and gradually to reveal information about an encrypted message, or about the encryption key itself.

2.2 Primitives

One of the primitive operations of cryptography is *encryption*. Let us consider two parties, a *sender* (Alice) and a *receiver* (Bob), who wish to communicate securely with each other via an insecure channel which is controlled by an *opponent* [1](Oscar). Encryption schemes can be divided into two categories: private-key (or symmetric) schemes, and public-key (or asymmetric) schemes. In a private key scheme, the same key is used for encryption and decryption. Hence, when two parties want to communicate securely with a symmetric encryption scheme, they need to exchange a private key in advance. In a public key scheme, two different keys are used for encryption and decryption. The key used for encryption, the public key, can be published, while the secret key used for decryption must be kept secret. An advantage of this is that, while the key for a symmetric encryption scheme must be exchanged securely, the public keys need to be exchanged using public channels, but the exchanged keys need to be authenticated. This is a much weaker requirement.

Both private key and public key encryption schemes consist of three algorithms: *key generation*, *encryption* and *decryption*. The difference between these two encryption schemes is reflected in the definition of security. The security of a public key encryption scheme should also hold when the adversary is given the encryption key, whereas this is not required for a private key encryption scheme because public key encryption schemes allow each user to broadcast his/her encryption key. Any user may send his/her encrypted messages to other users without agreeing on a private encryption key in advance.

[1]Throughout this thesis, we use the terms adversary, attacker, and enemy for the same meaning.

Models for evaluating security

The most important criterion to assess a cryptographic system is the security of the system. There are different security metrics by which a cryptosystem may be evaluated. Among them we consider the following:

- **Unconditional Security**

  A cryptosystem is said to be unconditionally secure if an adversary having unlimited computational resources cannot break the system. Unconditional security for an encryption system is called *perfect secrecy.* For perfect secrecy, observation of the ciphertext provides no information whatsoever to an adversary. A necessary condition for a symmetric-key encryption system to be unconditionally secure is that the key is at least as long as the message. The one-time pad [107] is an example of an unconditionally secure encryption algorithm. Public-key encryption schemes, however, cannot be unconditionally secure.

- **Computational Security**

  This measures the amount of computational effort required, by the best currently-known methods, to defeat a system. A proposed algorithm is said to be computationally secure if the perceived level of computation required to defeat it exceeds the computational resources of the hypothetical adversary.

  Most of the best known cryptosystems in current use are computationally secure. The members of this class are sometimes called *practically secure.*

- **Provable security**

  A cryptosystem is said to be provably secure if the difficulty of defeating it can be shown as equivalent (reduced) to the difficulty of solving a well-known difficult problem such as integer factorization [62] or the computation of discrete logarithms [117]. Provable security is considered to be adequate for most practical applications. Computational security includes provable security as a proper subset.

2.3 Private Key Cryptosystems

A private-key cryptosystem (i.e. symmetric-key encryption) enables two parties, the *sender* and the *receiver*, to communicate in secrecy via an insecure channel. Before any communication of messages takes place, both the sender and receiver must exchange the secret key $K \in \mathcal{K}$ via a *secure channel*, which can be implemented using a messenger or a registered mail. After exchanging the key, the sender can select a message $M \in \mathcal{M}$, apply the encryption algorithm $E : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$, and dispatch the cryptogram $C = E_K(M)$ through the insecure channel. The receiver, who knows the secret key K (we assume that the encryption/decryption algorithms are publicly known), recreates the message from the cryptogram using $D : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$, that is, $M = D_K(C)$.

The encryption system works correctly if

$$D_K\left(E_K(M)\right) = M$$

for all keys $K \in \mathcal{K}$. Data Encryption Standard (DES) algorithm [118], is a private-key cryptosystem which was developed at IBM in the mid 70s and used to be the standard encryption algorithm for many years. Rijndael [41] was selected recently as an advanced encryption standard (AES) to replace DES.

2.4 Public Key Cryptosystems

In private-key cryptosystems, both encryption and decryption keys are secret and either the same or the knowledge of one of them is sufficient to determine the other. The main drawback of applying private-key cryptosystems is that it requires the prior communication of the key K between sender and receiver, via a secure channel before any cryptogram is transmitted.

The main idea behind a public-key cryptosystem is that it might be possible to design a system that uses two different keys, e and d, for encryption and decryption respectively. The knowledge of one of these keys must not be sufficient (computationally) to determine the other. Hence, one of the keys can be published in a directory or public registry (this is where the name comes from).

Public key cryptography was invented in 1977 by Diffie and Hellman and was introduced in their paper entitled “New Directions in Cryptography” [56]. Their basic framework is as follows:

1. Find a computationally intractable problem Γ,

2. Build a cryptosystem based on Γ, in such a way that breaking the system is equivalent to solving instances of the intractable problem Γ.

Let $\{E_e : e \in \mathcal{K}\}$ be a set of encryption transformations and let $\{D_d : d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations (E_e, D_d). Suppose that each pair has the property that, knowing E_e and giving a random ciphertext $c \in \mathcal{C}$, it is computationally infeasible to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$. This property implies that, given e, it is infeasible to determine the corresponding decryption key d. E_e is viewed here as a trapdoor one-way function. The key d is the trapdoor information needed to compute the inverse function of E_e. Here a *trap-door one way function* means that it is easy to compute but hard to invert unless a trapdoor (the secret key) is known.

Under these assumptions, consider the two-party communication between Alice and Bob. Bob selects the key pair (e, d) and sends the encryption key e (called the public key) to Alice over any channel, but keeps the decryption key d (called the private key) secure and secret. Alice can subsequently send a message m to Bob by applying the encryption transformation determined by Bob's public key e to get $c = E_e(m)$. Bob can then decrypt the ciphertext c by applying the inverse transformation D_d uniquely determined by d. That is, $m = D_d(c)$.

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $E_e : e \in \mathcal{K}$ and $D_d : d \in \mathcal{K}$, respectively. The encryption method is said to be a *public key encryption scheme* if for each associated encryption/decryption key pair (e, d), one key, e (the public key), is made publicly available, while the other, d (the private key), is kept secret.

It appears that the public-key scheme (PKS) is an ideal system, not requiring a *secure channel* to pass the encryption key. This would imply that two entities could communicate over an unsecured channel without ever having met to exchange keys. Unfortunately, this is not the case. It can be shown how an active adversary can defeat the system (decrypt messages intended for a second entity) without breaking the encryption system. In this scenario, the adversary impersonates entity B by sending entity A a public key, e', which A assumes (incorrectly) to be the public key of B. The adversary intercepts encrypted messages from A to B, decrypts with its own private key, $\acute{d}$, re-encrypts the message under B's public key, e, and sends it on to B. This attack highlights the necessity to *authenticate* public keys to achieve data origin authentication of the public keys themselves. A must be convinced that he/she is encrypting under the legitimate public key of B.

2.4.1 The RSA Cryptosystem

Rivest, Shamir, and Adleman [132] were among the first to propose a concrete realisation of a trap-door one-way function as introduced by Diffie and Hellman[56]. It is based on the difficulty of computing the e-th root modulo a composite n, known as the RSA problem. Given a positive integer n, which is a product of two distinct odd primes p and q, a positive integer e such that $gcd(e,(p-1)(q-1)) = 1$, and an integer c, find an integer m such that $m^e = c \ (mod \ n)$. The RSA encryption scheme works as follows. Let p and q be two safe primes ($p = 2p' + 1$, $q = 2q' + 1$ and p' and q' are also primes), $n = pq$, and let e be an integer satisfying $gcd(e, \varphi(n)) = 1$, where $\varphi(n) = (p-1)(q-1)$. The public key of a recipient, Bob, is the pair (n, e). His secret key is the triple (p, q, d), where d satisfies $de = 1 \ (mod \ \varphi(n))$. To encrypt a message $m \in [0, ..., n-1]$ for Bob, a sender Alice computes

$$c = m^e \ (mod \ n)$$

and sends c to him. Bob can recover m using the secret value d as follows:

$$m = c^d \ (mod \ n).$$

The correctness of this encryption method is seen as follows:

$$c^d = (m^e)^d = m \ (mod \ n)$$

This holds since $ed = 1 \ (mod \ \varphi(n))$.

Security of RSA

The security of the scheme is based on the RSA problem. However, there are some pitfalls that can make the system insecure. For instance, when e is chosen as a small number for reasons of efficiency, a number of attacks are possible. Håstad showed that when encrypting the same message for multiple recipients having the same public exponent e but different modulo, the message can be computed from the cipher-texts without knowing any of the corresponding secret keys [80]. Furthermore, if polynomial relations among the encrypted messages are known, messages can also be recovered [39]. Such attacks can be prevented by *salting*, that is, appending a random bit-string to the message before encryption.

One obvious attack on the RSA cryptosystem is for a cryptanalyst to factorize the integer n (since knowing the factors p and q provides $\varphi(n)$ and hence the secret key

d). So, if the RSA cryptosystem is to be secure, it is necessary that n must be large enough so that factoring it will be computationally infeasible.

The RSA system is set up by receiver, Bob, who

- chooses two large and distinct primes p and q
- computes $n = p \times q$ and $\varphi(n) = (p-1)(q-1)$,
- selects at random the key $0 \leq e \leq \varphi(n)$, such that $gcd(e, \varphi(n)) = 1$
- computes the secret key d using the equation $ed = 1 \ (mod \ \varphi(n))$
- publishes n and e in a public directory registry as the public parameters of his RSA system.

Simmons and Noris [146] have shown that the private key d can be computed using their iteration attack (without factoring n),

2.4.2 The ElGamal Cryptosystem

Another notable public-key cryptosystem is the ElGamal [57]. This cryptosystem applies the discrete logarithm problem. (It can be seen as a special way of using the Diffie-Hellman key exchange protocol.) In this system, messages, cryptograms, and keys (public and secret) belong to a finite field $GF(p)$.

Let G be a finite cyclic group of order q and let $g \in G$ be a generator of G such that computing discrete logarithms in G is infeasible. In the original proposal, G was chosen to be Z_p^* (thus we have $ord(g) = p-1$) where p is a large prime. In order to encrypt a message $m \in G$ for Bob having the public key $e = g^d$, Alice first chooses r randomly in Z_q and computes the pair $(A, B) = (g^r, e^r m)$ being the cryptogram of m. Bob, knowing the secret key d, can receive the message m by calculating

$$\frac{B}{A^d} = \frac{y^r m}{g^{rd}} = \frac{g^{dr} m}{g^{rd}} = m$$

Alternatively, the role of the public key and the base can be interchanged and become the following variant. Now a message can be encrypted by randomly selecting an r in Z_q and computing the pair $(A, B) = (e^r, g^r m)$. Decryption is performed by calculating

$$\frac{B}{A^{d^{-1}}} = \frac{g^r m}{e^{rd^{-1}}} = \frac{g^r m}{g^r} = m$$

In both schemes, the security is based on the assumed intractability of the Diffie-Hellman problem. Note that these are probabilistic (non-deterministic) encryption schemes, that is, there are many valid ciphertexts for a given message. Furthermore, if a different r is used for every encryption, it is equivalent to the Decision Diffie-Hellman problem [17] to decide whether two pairs (A, B) and (A', B') both encrypt the same message m.

Security of ElGamal

One obvious attack on the ElGamal cryptosystem is for a cryptanalyst to obtain the random value r, using the first component of the cryptogram, and solve the discrete logarithm for g^r. If this can be done, it is simple to obtain the message m by computing e^r and applying its multiplicative inverse on the second component of the cryptogram. Hence, if the ElGamal cryptosystem is to be secure, it is necessary that p must be large enough that solving the discrete logarithm over $GF(p)$ is computationally infeasible.

Let p be a prime such that the discrete logarithm problem in Z_p is intractable, and let $0 \leq g \leq p-1$ be a primitive element. The following procedure shows how ElGamal system can be setup by receiver (Bob).

- Bob chooses (uniformly at random) the secret key d, $0 \leq d \leq p-1$
- he computes the public key, $e = g^d \ (mod\ p)$
- Bob publishes g, p and e in a public director as the parameters of his ElGamal cryptosystem.

ElGamal considered methods that a cryptanalyst may use to break his system [57]. As pointed out, all the attacks seem to be as difficult as the discrete logarithm problem. That is, the security of the ElGamal cryptosystem hinges on the difficulty of the discrete logarithm problem. Note that these conjectures are based on the assumption that the public parameters of the system are chosen properly. Bleichenbacher [24] has shown that if the public parameters are not chosen carefully, a particular type of attack is effective in forging an ElGamal signature. Since the secret key is not found in this attack, the difficulty of forging an ElGamal signature is sometimes weaker than the difficulty of the underlining discrete algorithm problem. Anderson and Vaudenay [9] have also discussed the effect of choosing improper public parameters.

Note. The ElGamal system is recommended to be used once only for any single integer r, $0 \leq r \leq p-1$. Every time Alice (or any one else) wants to send a message, she has to generate at random a new r. The violation of this requirement can be exploited by an opponent , say Oscar, who wants to decrypt a cryptogram knowing the message corresponding to another cryptogram. To illustrate this point, assume that Alice was careless and sent two cryptograms using the same r. Let them be: $C = (c_1, c_2) = (g^r, m_1 e^r)$ and $\hat{C} = (\hat{c_1}, \hat{c_2}) = (g^r, m_2 e^r)$. Oscar computes,

$$\frac{c_2}{\hat{c_2}} = \frac{m_1}{m_2}$$

which provides the opportunity to learn the message m_1 (or m_2) knowing the message m_2 (or m_1).

The ElGamal encryption algorithm requires two exponentiation, namely g^r $(mod\ p)$ and e^r $(mod\ p)$ (which is about two times of the RSA system). Although these exponentiations can be sped up by selecting random exponent r having some particular structure (e.g., having low Hamming weights), care must be taken that this does not make the system prone for any possible attack.

Another disadvantage of ElGamal encryption is that there is a message expansion by factor of 2. That is, the ciphertext is about twice as long as the corresponding plaintext.

2.5 Digital Signatures

Hand-written signatures are used in everyday situations such as signing a document or contract, or withdrawing money from a bank. Since a copy of a hand-written signature can usually be distinguished from an original, the signer cannot deny the original signature. This is why the signature is used to establish the responsibility of the signer for signed messages.

One of the greatest achievements of modern cryptography is the invention of digital signatures. Digital signatures should be in a sense similar to hand-written signatures. Since a copy of an electronic document is identical to the original, digital signatures have to create some sort of digital encapsulation for the document so that any interference with either its contents or the signature will be detected with a very high probability. In order to achieve this requirement, a digital signature on a message is a special encryption of the message that can be applied only by the legitimate signer. That is, in contrast to hand-written signatures, which are independent of the messages, the digital signatures must somehow bind to the message.

Of course, in both hand-written and digital signature schemes a third party, the receiver of the signature, must be able to verify the signature. A hand-written signature is verified by comparing it to other, authentic signatures. For example, in order to withdraw money from a bank, the bank compares the signature with one which is provided at the time when the account was opened. Verification of a digital signature, however, needs the application of a particular (in general, a publicly-known) algorithm. So, a digital signature scheme is a collection of three algorithms: a key generation algorithm, a signing algorithm, and a verification algorithm. They must have the following properties:

1. The key generation algorithm $Gen : \mathcal{K} \to e, d$ is used by each entity (signer and verifier) to generate large secret key d and its corresponding public key e.

2. The secret key is used for signing messages, and the public key is used by the other entities for verifying signatures.

3. The signing algorithm $Sig_d : \mathcal{K} \times \mathcal{M} \to \sum$ assigns a signature $\sigma = Sig_d(M)$, where $M \in \mathcal{M}$ is a message, $d \in \mathcal{K}$ is the secret key of the signer, and $\sum$ is the set of all possible values of the signatures.

4. The signing algorithm executes in polynomial time when the secret key d is known. For an opponent, who does not know the secret key, it should be computationally intractable to forge a signature, that is, to find a valid signature for a given message.

5. The verification algorithm $V_e : e \times \mathcal{M} \times \sum \to \{\text{yes, no}\}$ takes a public information $e \in \mathcal{K}$ of the signer, a message $M \in \mathcal{M}$ and a given signature $\sigma \in \sum$ of the message M. It returns "yes" if σ is the signature of the message M; otherwise it returns "no".

6. The verification algorithm, in general, is a publicly known (polynomial time) algorithm. Therefore, anyone can use it to check whether a message M matches the signature σ or not.

There are two main classes of digital signature schemes: digital signature schemes with appendix, and digital signature schemes with message recovery.

2.5.1 Digital Signature Schemes with Appendix

This class of digital signatures requires the original message as input to the verification algorithm. They rely on cryptographic hash functions and are the most commonly used in practice. An example of digital signature with appendix is the ElGamal [57] signature scheme.

ElGamal Signature

The ElGamal signature scheme has been the subject of investigation by several authors, and several variants of this signature have been introduced in the literature. In fact, the ElGamal system is designed specifically for the purpose of signatures, as opposed to the RSA system, which is suitable for both signature and other cryptographic purposes.

The ElGamal signature scheme is a non-deterministic signature scheme (like the encryption scheme), that is, there are many valid signatures for a given message, since the cryptogram depends on both the message and on the random value r chosen by the sender.

Thus, the verification algorithm must be able to accept any of the valid signatures as authentic. The description of the ElGamal signature scheme is as follows. As in the ElGamal cryptosystem, let d be the secret key and $e = g^d$ be the public key. Assume that Bob, the owner of the system, wishes to sign a message m , $m \in Z_p$ such that $gcd(r, p-1) = 1$, where r is randomly selected in Z_q and calculates

$$x = g^r \ (mod\ p)$$

Then, he solves the following congruences

$$M \equiv d.x + r.y(mod\ p)$$

for variable y. The signature of the message is

$$\sigma = Sig_d(M) = (x, y)$$

Upon reception of m and $\sigma = (x, y)$, Alice (or anyone else who knows the public key parameters of Bob's ElGamal cryptosystems) can verify Bob's signature using,

$$g^M \equiv e^x \times x^y$$

Note that possessing the pair (x, y) does not allow the message m to be recreated, that is, the ElGamal signature is a *digital signature with appendix.* In fact, there are many pairs which match the message: for every random value r there is a pair (x, y). The Schnorr scheme [137] is a well-known ElGamal-type digital signature scheme with appendix. However, Nyberg and Rueppel [116] is an ElGamal-type signature scheme but with message recovery.

2.5.2 Digital Signature Schemes with Message Recovery

Digital signature schemes with message recovery have the feature that the message signed can be recovered from the signature itself. In practice, this feature is of use for short messages. This class of digital signatures does not require the knowledge of the message for the verification algorithm. An example of digital signature with message recovery is the RSA [132] signature scheme.

RSA Signatures

In the RSA system, the signature algorithm is identical to the decryption. That is, to sign a message m $(0 \leq m \leq n)$ the owner of the secret key generates the signature using

$$\sigma = m^d \ (mod\ n)$$

The verification of the signature, however, is similar to the encryption. That is, everyone who the public key e can check the validity of (m, σ) using

$$(\sigma)^e \ = m \ (mod\ n)$$

If the above equation is satisfied, then the signature is accepted as a valid signature. Otherwise the signature is rejected as a forged one. Since the verification recovers the message signed, this sort of RSA signature is a it digital signature with message recovery

Note. As is seen, the signing algorithm produces a signature with the same length as the message. This is unsatisfactory as it needs double space for storage, and double bandwidth for transmission. Moreover, signing blocks may be subject to homomorphic attack [44] by which the message could be revealed. In order to avoid this problem, we shall assume, throughout this thesis, that a document (a message of arbitrary length) is first hashed, and the signature is then produced for its digest. Obviously, the hashing employed must be collision free and must avoid attacks which exploit existing algebraic structures in both the signing algorithm and the hashing function (see [51]).

2.5.3 One-time Signatures

One-time signatures are another class of digital signatures which can be used to sign, at most, one message; otherwise, signatures can be forged. A new public key is required for each message that is signed. One-time signatures were first proposed by

Lamport [97] and Rabin [128] based on the idea of committing to secret keys by one-way functions without trapdoors. This can be implemented using a fast hash function such as SHA-1 (see Section 2.6), resulting in magnitudes faster than traditional digital signatures. One-time signatures normally require the publishing of large amounts of data to authenticate messages because each signature can be used only once.

With the spread of low-powered, resource-constrained, small devices, such as the cell phone, pager, Palm pilot, and smart cards in recent years, one-time signatures have attracted more and more attention as an alternative to the traditional signature based on public-key cryptography.

Examples of other typical one-time signature schemes includes Merkle [108], GMR [66], Bos and Chaum [26]. The BiBa [121], "better than BiBa" scheme of Reyzin and Reyzin [129], HORS++ of [125], and $k-$Siblings of [7] are examples of some recent approaches in designing one-time signatures. In Section 7.3, we established a model for one-time signature schemes. The model is not aimed at introducing a new kind of signature, but to set a common view of several well-established signature schemes.

One-time signatures may tend to be unwieldy when used to authenticate multiple messages because additional data needs to be generated to both sign and verify each new message. By contrast, with conventional signature schemes like RSA, the same key pair can be used to authenticate multiple documents. There is a relatively efficient implementation of one-time-like signatures by Merkle [108] called the Merkle-Tree signature scheme which does not require new key pairs for each message.

2.5.4 Other Types of Digital Signatures

There are other types of special digital signatures in cryptography. They mainly involve more than one signer or more than one message to be signed.

The following table summarizes a number of "non-classical" signature schemes according to the number of participant signers and the number of messages to be signed.

Signer	**Messages**	**Schemes**	**Example**
1	n	Batch Signature	[60],[15]
$t \leq n$	1	Threshold Signature	[52],[63]
n	1	Multisignatures	[27],[35]
n	n	Concast Signature	[5]

Table 2.1: Special signature schemes

2.5.5 Performance of Digital Signatures

In almost all digital signatures, schemes generation/verification of a signature requires the performance of some exponentiation. For example, signature generation in ElGamal digital signature is relatively fast, requiring one modular exponentiation and two modular multiplications. However, signature verification is more costly, requiring three modular exponentiations and two modular multiplications. In contrast, RSA signature generation requires four modular exponentiations, but signature verification is significantly faster requiring few modular multiplications [107].

Since exponentiation is a costly operation, the design of efficient digital signature schemes (from both the generation and verification points of view) has been the subject of investigation by many researchers (see, for example, [61, 138, 55, 40]).

Currently, various applications use efficient forms of digital signatures to secure their applications [70]. Smart cards use digital signature which is implemented in the hardware as a silicon chip [72].

2.6 Hashing

There are several cryptographic applications which require the production of a short fingerprint (or a *digest*) of a much longer document/message. Cryptographic applications of hashing include, amongst others, the generation of digital signatures and message authentication. A hashing function h, in general, is a procedure that takes as input a message M of arbitrary length and produces a digest $h(M)$ of a fixed length.

2.6.1 Properties

In order to assess the security of a hash function, a commonly used criterion is the collision freeness property. A hash function h is called *collision free* if finding messages M_1 and M_2 with $h(M_1) = h(M_2)$ is a hard problem [42]. A formal definition of a collision free, also called strong one-way, hash function h is given as follows:

- h can be applied to any message or document, M, of any size.
- h produces a fixed size digest $h(M)$.
- *preimage resistant.* Given h and M, it is easy to compute $h(M)$, but it is computationally intractable to find the message M for the given digest $h(M)$, that is, h is one-way.

- *2nd preimage resistant.* Given the description of the hash function h, and a chosen message M, it is computationally intractable to find another messages $\dot{M}$ which collides with M, i.e., $h(M) = h(\dot{M})$. That is, h is collision free [2]. $2nd$ preimage resistance is also equivalently termed *weak collision resistance.*

There are two major classes of hash function defined as follows:

1. *A one-way hash function* (OWHF) compresses messages of arbitrary length into digests of fixed length. The function is preimage and $2nd$ preimage resistant. Equivalently, the function is termed *weak one-way hash function.*

2. *A collision resistant hash function* (CRHF) is OWHF with additional properties: collision resistant and $2nd$ preimage resistant. Equivalently, the function is termed *strong one-way hash function.*

Several constructions of hash functions (for different purposes and with different levels of security) have been proposed in the literature (see, for example, [136], [168] and [124]).

One important concept introduced by Naor and Yung [114] is the Universal One-way Hash Function (UOWHF) which comprises a collection of hash functions where based on a probabilistic polynomial time algorithm that can determine and return a hash function h on a specific argument. By definition: let H be a polynomial computable and accessible hash function compressing $\ell(n)$-bit input into n-output strings and $\mathcal{F}$ be a collision finder. H is a universal one-way hash function if for each $\mathcal{F}$, for each polynomial Q, and for all sufficiently large n

$$P(\mathcal{F}(x, h) \neq ?) < \frac{\infty}{Q(n)}$$

where $x \in \mathit{\Sigma}^{\ell(n)}$ and $h \in_{\mathcal{R}} H_n$. The probability is computed over all $h \in_{\mathcal{R}} H_n$, $x \in \Sigma^{\ell(n)}$ and the random choice of all finite strings that $\mathcal{F}$ could have been chosen.

The main difference between UOWHF and OWHF is the way the hash function is chosen. In the case of OWHF, the hash function is fixed. For UOWHF, the hash function is randomly chosen. Note that UOWHF is 2nd preimage resistant.

Naor and Yung demonstrated how to construct a family of UOWHF by the composition of one-way permutation and a family of strongly universal hash functions with collision accessibility property. Rompel [134] proved that universal one-way hash function can be constructed from any one-way function.

[2]There are, obviously, many collisions for a hash function h, since the message source is much larger than the digest source.

Sibling Hashing

In some applications, it may be useful to have a hash scheme with an easy to find collection of colliding messages. The calculation of other collisions should be computationally intractable. This can be achieved using a particular type of hash functions called SIFF. The sibling intractable hash function (SIFF) has the property that given a set of initial colliding strings, it is computationally infeasible to find another string that would collide with the initial strings. It is constructed from applying two types of functions: universal hash function and collision resistant hash function [170].

MD algorithms

MD2 [86], MD4 [130], and MD5 [131] are message-digest algorithms developed by Rivest. They are meant for digital signature applications where a large message has to be "compressed" in a secure manner before being signed with the private key. All three algorithms take a message of arbitrary length and produce a 128-bit message digest. While the structures of these algorithms are somewhat similar, the design of MD2 is quite different from that of MD4 and MD5. MD2 was optimized for 8-bit machines whereas MD4 and MD5 were aimed at 32-bit machines.

The Secure Hash Algorithm (SHA), the algorithm specified in the Secure Hash Standard (SHS), was developed by NIST. SHA-1 [115] was a revision to SHA that was published in 1994. Its design is very similar to the MD4 family of hash functions. The algorithm takes a message of less than 264 bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute-force collision and inversion attacks.

It is worth mentioning that the MD family is no longer secure. SHA-1 is widely acceptable and proved to be secure so far.

2.6.2 Keyed Hashing

A *message authentication code* (MAC) is a relatively short string which is attached to a message to enable the receiver to decide whether the message comes from the original sender. Clearly, to perform this role, the MAC must match the message and the sender. As the message can be long and the MAC is relatively short, it must directly or indirectly employ hashing. Additionally, the pair of communicating parties is uniquely identified by a secret key shared by them. To produce or verify a MAC, the parties must know the message m and the shared key. An adversary, on the other hand, knows the message pair (m,MAC) only. Applications of MACs allow the creation of an authentication channel where the contents of the message is public but the message source can be verified (the sender and the receiver must share the same key). Keyed hash schemes produce digests which depend on not only messages but also secret keys which are shared between the sender and the receiver. Consequently, hashing can be done only by the holders of the secret key [124].

Given a family of hash functions

$$\bar{H}_n = \{h_k : \Sigma^* \to \Sigma^n \mid k \in \Sigma^n\},$$

any instance function h_k is indexed by a secret key k shared between two parties. A *keyed hash function* $H = \bar{H}_n \mid n \in \mathcal{N}$ is collision resistant if, for each n:

1. any instance function h_k can be applied for messages of arbitrary length,
2. the function H is a *trapdoor one-way* function, that is
   - given a key k and message m, it is easy (in polynomial time) to compute the digest $d = h_k(m)$
   - for any polynomial size collection of pairs $(m_i, d_i = h_k(m_i))$; $i = 1, \ldots, \ell(n)$, it is intractable to find the key $k \in \sum^n$, where $\ell(n)$ is a polynomial in n.
3. without the knowledge of k, it is computationally difficult to find a collision, that is, two distinct messages $m, m' \in \sum^*$ with the same digest $d = h_k(m) = h_k(m')$.

For an ideal collision resistant keyed hashing scheme, finding a collision could be done either by applying an exhaustive search through the key space, which takes on the average 2^{n-1} operations, or by employing a variant of the birthday attack, which takes $O(2^{n/2})$ steps.

Attacks on Hashing Functions

There are many methods of attack on hash schemes. The so-called *birthday attack* is a general method which can be applied against any type of hash function. Some others are special attacks such as the so called *meet-in-the-middle attack* that can be launched against any scheme that uses block chaining. Others can only be launched against smaller groups.

The *birthday paradox* is stated as follows: what is the minimum number of pupils in a classroom so the probability that at least two pupils have the same birthday is greater than 0.5? The answer is 23. This fact arises when drawing elements randomly, with replacement, from a set of N elements, with high probability a repeated element will be encountered after $O(\sqrt{N})$. Equivalently, the birthday paradox can be employed to attack hash functions, that is, it is easier to find collisions for a one-way hash function than to find a pre-image of specific hash values.

2.6.3 Digital Signatures vs. Message Authentication Codes

Both signature schemes and message authentication codes are methods for validating data, that is, verifying that the data was approved by a certain party (or set of parties). The difference between signatures and message authentication is that signatures should be universally verifiable, whereas authentication codes are only required to be verifiable by parties that are also able to generate them.

The difference between signatures and message authentication codes is captured by the security definition, and affects the possible applications of these schemes. In case of message authentication, the verification-key is assumed to be kept secret (and so these schemes are "private-key" type), whereas in the case of signature schemes the verification key may be made public (and so these schemes are of the "public" type).

The difference between the two authentication methods arises from the difference in the settings for which they are intended, which amounts to a difference in the identity of the receivers and in the level of trust that the sender has in the receiver. Typically, message authentication schemes are employed in cases where the receiver is predetermined and is fully trusted by the sender, whereas signature schemes allow validation of the authenticity of the data by anybody.

2.7 Secret Sharing Schemes

One of the fundamental concepts and techniques in cryptography is secret sharing. This section will give the basic concepts of threshold secret sharing schemes which is used later in the thesis.

2.7.1 Basic Concepts

A simple example can be used to illustrate the concept of secret sharing. For instance, in a bank there is a vault that will be opened every day by three senior tellers but a single teller is not to be trusted to open it. So the bank has to decide how to open it safely. They would like to have a way in which any two of the tellers can open the vault but no single teller can open it. The system that allows this is (2,3) threshold secret sharing scheme.

Assume that there is a key set $\mathcal{K}$ and the key $K \in \mathcal{K}$, the set $\mathcal{S}$ is the set of all possible values of the shares, and $\mathcal{P}$ is the set of n principals. There is also a dealer D, $D \notin \mathcal{P}$, who chooses the key K and assigns the shares among the n principals. If the number of principals in B is equal to or greater than t, $|\mathcal{S}| \geq t$ and $\mathcal{S} < \mathcal{P}$, principals in $\mathcal{S}$ can find out the key K. If less than t, principals cannot find any information of the key K.

Formally, let t, n be positive integers, $t \leq n$. A (t, n)-*threshold scheme* is a method of sharing a key K among a set of n principals (denoted by $\mathcal{P}$), in such a way that any t principals can compute the value of K but no group of $t - 1$ principals can do so.

Threshold secret sharing schemes were first introduced, independently, by Shamir [139] and Blakley [22]. Shamir and Blakley schemes have been widely investigated in the literature (for example, Asmuth and Bloom [10], Karnin *et al.* [87], Mignotte [109], Kothari [93], Blakley and Meadows [23], Meadows [106], De Soete and Vedder [149], Stinson and Vanstone [155], Laih *et al.* [96], Simmons [144, 143] and Blakley *et al.* [21]). For a survey of different schemes, refer to Simmons [145] and Stinson [154].

A secret sharing scheme may have the following properties:

1. *Perfectness:* A secret sharing scheme is called *perfect* if all subsets of principals that do not form a qualified set are unable to obtain any information about the secret or about shares of other principals. Such schemes are also called information theoretically secure. There are also computationally secure secret sharing schemes, where it is infeasible to compute the secret for any subset not in a qualified set (e.g., [33]).

2. *Ideal*: A perfect secret sharing scheme is called *ideal* if the size of the shares equals the size of the secret.

3. *Verifiable*: A secret sharing scheme is called *verifiable* if each principal can verify that he/she has indeed obtained a valid share, that is, the dealer need not be trusted. Such a scheme was first proposed by Chor *et al* [36]. Verifiable secret sharing schemes require a third algorithm *ver*, that takes as input a share, and outputs true if and only if the share is valid. Hence, the principals can convince themselves that the shares are valid. Schemes that also allow other entities to validate the shares of all principals are called *publicly verifiable* [150].

2.7.2 Shamir Threshold Scheme

There follows a brief review of the Shamir threshold scheme [139]. The Shamir (t, n) threshold scheme is based on polynomial interpolation. Given t points in the two-dimensional plane $(x_1, y_1), \ldots, (x_t, y_t)$ with distinct x_i's, there is one and only one polynomial $f(x)$ of degree at most $t-1$ such that $y_i = f(x_i)$ for all i. The polynomial is given by the Lagrange interpolation formula as follows:

$$f(x) = \sum_{i=1}^{t} y_i \prod_{\substack{j=1 \\ j \neq i}}^{t} \frac{x - x_j}{x_i - x_j}. \tag{2.1}$$

Let the secret be an element of a finite field, that is, $K \in GF(p)$, where p is a prime number. Since polynomial interpolation is possible over $GF(p)$, Shamir suggests the following algorithm for constructing a (t, n) threshold scheme.

Set-up Phase:

1. The dealer, D, chooses n distinct non-zero elements of Z_p, denoted $x_1, \ldots, x_n$ and sends x_i to P_i via a public channel.
2. D secretly chooses (independently and randomly) $t-1$ elements of Z_p, denoted $a_1, \ldots, a_{t-1}$ and forms the polynomial
$$f(x) = K + \sum_{i=1}^{t-1} a_i x^i.$$
3. For $1 \leq i \leq n$, the dealer computes s_i, where
$$s_i = f(x_i) \pmod{p}.$$
4. D gives (in private) share s_i to principal P_i.

Secret Reconstruction Phase:

Every subset of $\mathcal{P}$, which has at least t principals can apply the Lagrange interpolation formula to reconstruct the polynomial and hence to recover the secret.

The principals do not need to reconstruct the polynomial $f(x)$. The secret is the constant term of the polynomial, that is, $K = f(0)$. So, they can recover the secret using:

$$K = \sum_{j=1}^{t} s_{ij} \prod_{\substack{k=1 \\ k \neq j}}^{t} \frac{x_{ik}}{x_{ik} - x_{ij}} \pmod{p}. \tag{2.2}$$

An alternative method of secret reconstruction is to solve linear equations in Z_p. Every set of at least t principals can always form the following system of equations:

$$\begin{aligned} K + a_1 x_{i1} + a_2 x_{i1}^2 + \cdots + a_{t-1} x_{i1}^{t-1} &= s_{i1} \\ K + a_1 x_{i2} + a_2 x_{i2}^2 + \cdots + a_{t-1} x_{i2}^{t-1} &= s_{i2} \\ \vdots \qquad\qquad & \quad \vdots \\ K + a_1 x_{it} + a_2 x_{it}^2 + \cdots + a_{t-1} x_{it}^{t-1} &= s_{it} \end{aligned}$$

This can be written as:

$$\begin{pmatrix} 1 & x_{i1} & x_{i1}^2 & \cdots & x_{i1}^{t-1} \\ 1 & x_{i2} & x_{i2}^2 & \cdots & x_{i2}^{t-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{it} & x_{it}^2 & \cdots & x_{it}^{t-1} \end{pmatrix} \begin{pmatrix} K \\ a_1 \\ \vdots \\ a_{t-1} \end{pmatrix} = \begin{pmatrix} s_{i1} \\ s_{i2} \\ \vdots \\ s_{it} \end{pmatrix}.$$

The leftmost matrix is a so-called Vandermonde matrix and its determinant $det\mathcal{A}$ is given by the following formula:

$$\det \mathcal{A} \equiv \prod_{1 \le k < j \le t} (x_{ij} - x_{ik}).$$

Since all x_i's are distinct, no term $(x_{ij} - x_{ik})$ is zero. Thus the determinant of a Vandermonde matrix over a finite field is always non-zero and the above system of equations has a unique solution over Z_p. That is, every set of at least t principals can uniquely reconstruct the polynomial and hence recover the secret.

Note. The size of $GF(p)$ must be large enough such that the selection of distinct and non-zero elements x_i's is possible. That is, the required condition for constructing a Shamir (t, n) threshold scheme is that the prime number p (the size of the field) must be greater than n (the number of principals in the system).

The computational cost of a secret sharing scheme is in solving a set of linear systems of degree t, which is a polynomial time complexity problem. The Shamir threshold scheme is an example of an ideal secret sharing scheme.

2.7.3 Verifiable Secret Sharing Scheme

One problem of secret sharing schemes is that they are not secure against cheating principals who send false shares when the secret is to be recovered. Another problem is that a cheating dealer could distribute false shares, so that different groups of principals recover different secrets. Such problems arise in protocols for secure multi-party computations (see e.g., [19]), and can be solved with verifiable secret sharing (VSS) schemes [36].

The object of verifiable secret sharing (VSS) is to resist malicious principals who are:

1. A dealer sending incorrect shares to some or all of the principals, and

2. Principals submitting incorrect shares during the reconstruction stage.

A VSS scheme is a secret sharing scheme with an additional, interactive or non-interactive algorithm which allows the principals to verify the validity of their shares. In other words, all groups of principals recover the same value if their shares are valid and this unique value is the secret if the dealer was honest.

In a VSS scheme, the principals can verify the validity of only their own share, but they cannot know whether other principals (with whom they might be able to recover the secret) have also received valid shares. This problem can be solved with publicly verifiable secret sharing (PVSS).

Model for PVSS

A typical secret sharing scheme consists of a *dealer*, n, principals $P_1, \cdots, P_n$, and an access structure $\mathcal{A} \subseteq 2^{\{1,\cdots,n\}}$. The access structure is monotone, which means that if $A \in \mathcal{A}$ and $A \subseteq B$ then $B \in \mathcal{A}$. For instance, in a threshold secret sharing scheme with threshold k, the access structure is defined as $\mathcal{A} = \{A \in 2^{\{1,\cdots,n\}} || A| \geq k\}$, which means that any coalition of at least k principals can recover the secret.

To share a secret s among the principals, the dealer runs an algorithm *Share*

$$Share(s) = (s_1, \cdots, s_n)$$

to compute the shares. The dealer then sends each share s_i secretly to $P_i, i = 1, \cdots, n$. If a group of principals wants to recover the secret, they run an algorithm *Recover*, which has the property that

$$\forall A \in \mathcal{A} : Recover(\{s_i | i \in A\}) = s,$$

and that for all $A \notin \mathcal{A}$ it is computationally infeasible to calculate s from $\{s_i | i \in A\}$. Thus, only those coalitions of principals belonging to the access structure $\mathcal{A}$ are able to recover the secret s.

In a VSS scheme, an additional, possibly interactive algorithm *Verify* allows the principals to verify the validity of their shares:

$$\exists u \forall A \in \mathcal{A} : (\forall i \in A : Verify(s_i) = 1) \Longrightarrow Recover(\{s_i | i \in A\} = u),$$

and $u = s$ if the dealer is honest.

In a PVSS scheme, a public encryption function E_i is assigned to each principal P_i, such that only he knows the corresponding decryption function D_i. The dealer now uses the public encryption functions to distribute the shares by calculating

$$S_i = E_i(s_i), i = 1, \cdots, n$$

and publishing the encrypted shares S_i. To verify the validity of all the encrypted shares, there is an algorithm *PubVerify* with the property that

$$\exists u \forall A \in 2^{\{1, \cdots, n\}} :$$

$$(\mathit{PubVerify}(\{S_i | i \in A\}) = 1) \Rightarrow \mathit{Recover}(\{D_i(S_i) | i \in A\}) = u$$

and $u = s$ if the dealer is honest. In other words, if a set of encrypted shares is "good" according to *PubVerify*, then the honest principals can decrypt them and recover the secret. Note that *PubVerify* can be executed even if the principals have not received their shares so far. To run *PubVerify*, it may be necessary to communicate with the dealer (but not with any principal). A PVSS scheme is called non-interactive if *PubVerify* requires no interaction with the dealer at all.

2.8 Cryptographic Protocols

A *cryptographic protocol* or *cryptographic scheme* is an algorithm defined by a sequence of steps precisely specifying the actions required for two or more entities to achieve a specific objective (see [107]).

Protocols play a major role in cryptography and are essential in meeting cryptographic goals such as privacy, confidentiality, data integrity, entity authentication or identification, message authentication, signature and authorization.

A *protocol failure* occurs when a mechanism fails to meet the goals for which it was intended, in a manner whereby an adversary gains advantage not necessarily by breaking an underlying primitive such as an encryption algorithm directly, but by manipulating the protocol or mechanism itself [107].

There are many reasons that can cause the failure of protocols, examples are:

1. Weaknesses in a particular cryptographic primitive which may be amplified by the protocol or mechanism;

2. Claimed or assumed security guarantees which are overstated or not clearly understood; and

3. The oversight of some principle applicable to a broad class of primitives, such as encryption.

When designing cryptographic protocols and schemes, the following two steps are essential:

1. Identification of all assumptions in the protocol or mechanism design; and

2. For each assumption, determination of the effect on the security objective if that assumption is violated.

A commonly accepted methodology for analyzing the security of cryptographic protocols consists of the following two steps. One first designs an ideal system in which all parties (including adversary) have oracle access to a truly random function, and then proves the security of this ideal system. Next, one replaces the random oracle by "good cryptocraphic hashing function" providing all parties (including the adversary) with succinct description of this function. Thus, one obtains an implementation of this ideal system in a world where random oracle does not exist. This methodology was explicitly formulated in [17].

Chapter 3

Multicast: Structure and Security

Multicast has become an important mode of communication as well as a good platform for building group-oriented services. However, to be used for trusted communication, current multicast schemes must be supplemented by mechanisms for protecting traffic, controlling participation, and restricting access of unauthorized users to data exchanged by the participants. In this chapter, we first review the fundamentals of multicast technology, and then we consider the issues and mechanisms for building a secure multicast environment. We survey the group-key management and distribution protocols since they are the corner-stone for building a multicast security architecture.

3.1 Multicast Illustrated

There are three fundamental types of communication modes in the internet: unicast, broadcast, and multicast. While unicast is a point-to-point communication, broadcast is a one-to-all communication, multicast is a one-to-group communication. Multicast is a relatively new communication mode and it is becoming an important tool in networking communications. It has become an essential tool particularly with the exponential growth of internet users, with the emergence of group-based applications, and with the extensive use of the Mbone (Multicast Backbone of the Internet).

Multicasting Advantages

Multicasting delivers high-bandwidth applications without overloading networks. As a result, it is ideal for distributing multimedia content across intranets and the Internet. Multicasting makes implementing hungry-bandwidth applications such as push-technology and video server applications without concern about traffic bottlenecks.

Multicasting benefits organizations where large numbers of individuals must receive the same information. At the same time, it is a suitable solution for communicating real-time information to large audiences, including sales and press conferences, because it sends the information only once, to multiple recipients. This is an efficient way of saving bandwidth and maintaining scalability of the service.

IP multicasting enhances the organization network by enabling a wide variety of applications, and by improving productivity and competitiveness of an enterprise. Multicasting is an ideal infrastructure for collaborative high bandwidth applications. Examples of such applications are teleconferencing, news feeds, pay-TV, on-line video, live stock quotes, Internet games, shared white-boarding, webcasting and many others.

The immediate advantage of multicasting is reducing bandwidth requirements. The sender is not required to transmit as many sequential or concurrent packets and is available for other tasks. To appreciate this, consider a host that wants to send a message to a group of n users. If unicast communication is used, then n different unicast connections need to be established between the sender and the receivers. Using multicast communication, the host would only need to send the message once, and the multicast infrastructure would duplicate the message to the rest of the group members in an efficient way. Yet, if n users of the group want to communicate with each other, then an $n \times n$ unicast connection is needed. The other advantage of multicast communication is that it can save alot of network resources in the already congested environment.

Multicast is becoming more demanded in the IP networks and, at the same time, more challenging to be applied in other types of networks such as Wireless, mobile, and ATM networks [140]. Multicasting does, however, require some changes in the networking infrastructure and desktops, including selecting special routing protocols, and setting up switching and securing intranets.

The only difference between a multicast and unicast IP packet, in terms of addressing, is the presence of a 'group address' in the the destination address field of the IP header. Instead of Class A, B, or C IP destination address, multicasting employs a Class D address format, which ranges from 224.0.0.0 to 239.255.255.255.

Multicasting is not a connection-oriented mode of communication. IP is a connectionless environment (*lossy*) and a multicast datagram is delivered to destination group members with the same "best effort" reliability as a standard unicast IP datagram. This means that multicast datagrams are not guaranteed to reach all members of a group, nor to arrive in the same order in which they were transmitted [156].

How Multicasting Works

Multicasting is typically formed by creating a group where participants place their information which is destined to all other participants of the group. This group can be in the form of a newspaper, IP address or Asynchronous Transfer Mode (ATM) address.

Membership to a multicast group is often highly dynamic, with receivers entering and leaving the multicast session without the permission or explicit knowledge of other hosts. Hosts join a multicast group using IGMP [46, 59, 30]. Once a host joins the group, it can exchange information with the rest of the group members.

As with unicast communication mode, multicast also requires a router for the transportability of information through the network. Multicast routers require a multicast routing protocol such as MOSPF [111], DVMRP [159], or PIM [50]. The mechanism used by the multicast in the Internet is to construct a multicast tree at the network layer that spans to all group members. The key issue is to have an efficient routing protocol that enables the transmission of the packets from a sender to a group of receivers. Many of the multicast protocols were developed to ensure efficient routing of multicast data. The objective of the routing protocols is to create an optimum multicast delivery path through the network. These routing protocols use a variety of algorithms to achieve the most efficient delivery process. The efficiency is measured by the shortest available path only to those clients who want to receive it.

Additional Multicast Requirement

The additional requirement includes secure data transmission to recipients to ensure both confidentiality and integrity of the messages. A number of security protocols have also been developed to ensure secure multicast group communication. Other studies have focused on reliability [104].

The purpose of this chapter is first to give an evolutionary survey of the multicast routing protocols, and then to give a brief survey of multicast security proposals in the literature.

3.2 Multicast Evolution

Multicast technology has been developed based on a number of techniques and protocols that formed the story of its evolution [120, 103].

RPF

The starting point was the development of the Reverse Path Forwarding (RPF) algorithm which solves the problem of looping in broadcast networks. The problem is caused by a sender sending a packet to multiple recipients, where there is a chance for intermediate routers to receive a duplicate of the same packet from neighboring routers. The rule adopted at a router is to accept packets that only arrive from the interface that it received the packets from. Although RPF solves the problem of looping, it did not solve the problem of group membership. Therefore, RPF was considered as a technique used to build a broadcast tree rather than a multicast tree [20].

IGMP

The next requirement was in developing a method to allow a participant to join a communication group. Internet Group Multicast Protocol (IGMP) is a protocol used between a host on a subnet and the corresponding router that it wants to join in a particular IP multicast group. Usually, the router advertises for a group membership by sending queries to hosts of the subnet and expecting replies from the host requesting membership to the group. When a host is interested in joining a particular multicast group, it uses the IGMP protocol to join the group. When a host leaves a group, the router stops sending data to it and frees bandwidth on the network segment of the host. IGMP is a standard TCP/IP protocol. IGMP was the result of the pioneer work of Deering [47] in defining multicast and its requirement in the late 1980s. Two versions of this protocol were proposed [46, 59], and the third is under standardization [30].

TRPB

The group membership IGMP was augmented with RPF to provide Truncated Reverse Path Broadcasting protocol (TRPB), to enjoy the best of the two features of IGMP and RPF: building a loop free multicast. In this scheme, the membership of hosts is taken into account when deciding to broadcast packets to a group. If there are no members of a group in the subnet, then the router will not forward the packet. While this technique reduce the traffic on the leaf subnet, it does not reduce traffic on the core network.

DVMRP

With the rise of group membership protocol and robustness of unicast network layer routing algorithms, the first multicast routing protocol was developed. Distance Vector Multicast Routing Protocol (DVMRP) refines truncated broadcasting by using a mechanism called *pruning* [159]. DVMRP is based on the network layer link state routing algorithm in which the router computes the shortest path to destinations based on information gathered from the advertisement of the adjacent routers. When a router gets the flooded packets, it knows if it is useful for it or not. That is, if there are no group members on the subnet, it sends a prune message unsteadily towards the sender. In this case, the unwanted branches of the spanning tree are pruned off. DVMRP is soft state, that is, the state of the router regarding the group membership is timed-out, and new flooding will allow new hosts requiring membership to graft the membership.

MOSPF

While DVMRP was based on the unicast routing protocol RIP [81] to compute the shortest path, Multicast Open Shortest Path First (MOSPF) [111] is another multicast routing protocol that is based on the link-state unicast routing protocol OSPF [112, 113]. OSPF advertises the state of its directly attached links and, based on these advertisements, each router builds its database. The MOSPF-enabled routers maintain multicast group membership information in addition to topological information. Based on these two pieces of information, MOSPF routers compute the shortest path from the sender to the group members using Dijkstra's shortest path algorithm. MOSPF uses two techniques: one for intra-area multicasting and another for inter-multicasting. Both use extra necessary extensions.

CBT

Core Based Tree (CBT) is another interesting multicast network-layer multicast routing protocol, which uses a single bi-directional, shared tree for a group [13, 11]. This is in contrast to several other protocols, such as DVMRP, which rely on a source-based shortest path tree. Theoretically, DVMRP creates as many delivery trees as the number of senders. However, there is a single delivery tree in CBT regardless of the number of senders. The main advantage of using a single shared tree per group is the reduction of state information that needs to be maintained at each router. For example, in a scheme like DVMRP, which uses source-based shortest path trees, a router may need to maintain as many as n entries of the form (S_i, G) for $i = 1$ to n where S_i is the source address, G is the multicast-group address, and n is the number of senders. In CBT, on the other hand, a router needs to maintain only a single shared entry irrespective of number of senders. This is a scalable advantage in CBT because the greater the number of routing table entries, the longer is the search time for finding the correct outgoing interface. As the number of multicast sessions increases, the performance of the router will decrease. The second advantage of CBT is saving communication bandwidth. Since flood and prune of DVMRP leads to a tremendous waste of network resources, CBT is designed to setup explicit join and leave messages from the members of the multicast session. A CBT consists of a core router, a Designed Router (DR) that is directly attached to the hosts, and intermediate routers along the branches of CBT from DR to the core. CBT protocol is distinguished by its provision of its own protocol header.

PIM

Protocol Independent Multicast (PIM), unlike other multicast routing protocols such as DVMRP or MOSPF, does not depend on any particular unicast routing protocol. PIM can cooperate with any unicast routing protocol. PIM has two modes: dense mode and sparse mode. Dense modes are designed for networks densely populated with members of a multicast group, while sparse mode is designed for network sparsely populated networks. The dense mode is the same as DVMRP in that it makes sense to flood the data to host, but the difference is that PIM does not depend on any unicast routing protocol. PIM has its unique architecture [50, 48, 49].

Mbone

To do IP multicasting on the Internet, all routers in the Internet have to be *multicast capable*, that is, each router needs to support multicast protocols such as DVMRP, MOSPF, PIM or CBT. In addition, the routers connected to subnets need to support IGMP. However, many of the existing routers on the Internet do not support any multicast routing protocol.

This leads to a chicken-and-egg problem in the sense that, on one hand, IP multicast cannot be deployed without the support of router vendors, while on the other hand, the router vendor would not support IP multicast before the technology is robust.

The circular dependency was broken by building multicast-capable subnets at the periphery of the Internet, and interconnecting them using IP tunnels. The main idea is to transmit IP multicast packets on the MBone encapsulated in regular IP packets so that they look like normal unicast IP datagrams to the intervening routers. The encapsulation is done in such a way that the tunnel endpoint receiving the encapsulated packet knows that the payload of the IP packet is another IP packet. The tunnel endpoint will then strip off the packet and forward the packet to local host, or re-encapsulate again to the next hop.

3.3 Multicast Security

Securing multicast networks has become a lively area of research and a challenging topic. The security challenge for multicasting is in providing an effective method of controlling access to the group resources that is as efficient as the underlying multicast. There are many different ways of looking at multicast security, and many different security requirements depending on the application.

To use multicast for trusted communication, the multicast architecture should be supplemented with mechanisms for protecting traffic, controlling participation, and restricting access of unauthorized users to the data exchanged by the participants. In other words, what is needed is a security architecture for multicast infrastructure.

Potential security threats to multicast communications are similar to those encountered in unicast transmissions. An adversary may eavesdrop on confidential communication, disrupt or distort a session data exchange, inject unauthorized or bogus traffic, block a session progress, masquerade as someone else to join in a session or initiate and operate a bogus session. However, because of the inherent broad scope of a multicast session, the potential for attacks may be greater than for unicast traffic.

The nature of multicast presents new problems that cannot be effectively dealt with using trivial extensions of techniques for secure unicast. For example, in unicast communications, peers know the identity of the other part. In a dynamic multicast session, on the other hand, knowing who are present within the session is typically not guaranteed. Furthermore, the number of communication links traversed by wide-area multicast can potentially be far greater compared with a single unicast, where the communication path is a collection of links and nodes between just one source and one destination. Therefore, multicast intrinsically creates more opportunity for traffic intercepting.

The easiest way to ensure security in a group-oriented communication is to establish $O(n^2)$ secure unicast channels using IPSec [88, 89, 90]. However, this loses the advantage and the flavor of the multicast facility over unicast, and has additional overhead. Also, there is no security control over who can register a session or who can join or leave a session. From the above argument, one can appreciate the need for a trusted multicast architecture to handle these security issues in a coherent manner and to protect network services and applications [68, 95, 12]. Therefore, we conclude that multicast communication is at a substantially increased risk and complexity compared to unicast communication.

In the study of multicast security, one can establish a taxonomy of the distinguishing security concerns, which can be listed in the following categories:

1. Multicast group characteristics: what the characteristics of the participant members as well as their data are .
2. Group Key management: or how to distribute the group key securely to group members in an efficient and secure manner.
3. Source Authentication: how to make sure of the identity of a participant host from which the information originated.

It is worth mentioning another low-level but vital security issue in multicast, which will not be discussed in detail: the *security of multicast routing.* Routing protocols dynamically configure the packet forwarding function in a network, which allows for the continued delivery of packets in spite of changes in network topology. These changes typically occur due to the ongoing emerging, failure, and repair of network links and routing nodes which the protocols have been designed to accommodate. The compromise of the routing function in a network can lead to the denial of service attack, the disclosure of network traffic, or the inaccurate accounting of network resources usage. Current routing protocols, including multicast routing protocols such as MOSPF, DVMRP, CBT and PIM, contain few, if any, mechanism to provide security for their operation. Securing routing protocols has been the subject of some studies [69, 141, 148, 147].

3.3.1 Multicast group characteristics

There are a number of parameters that characterize a multicast group. These parameters crucially affect which security architecture should be used [31]. For example:

- *group size* can vary from a few to thousands of participants

- *member characteristics* can vary from pocket to mainframe computers
- *member joining* to multicast group can be static or dynamic
- *membership control center* that has information about group membership, and the life time of the group
- *number and type of senders* can be single or multiple parties sending data
- *volume and type of traffic* can be heavy traffic and with real-time arrival such as online application

3.3.2 Group key management and distribution

Key management is the most challenging multicast security problem. The goal of key management is to distribute the group key securely to the group members, who can then use it to encrypt or decrypt the multicast data. The advantage of having a group key is that a sender avoids having to encipher traffic individually for each receiver. Key management deals with issues like bandwidth scalability, the number of key messages exchanged with increasing group size, initializing the multicast group members with group key, and re-keying the group members in case of joining or leaving. Specifically, it deals with answers to the following problems:

- How to authenticate potential group members
- How to distribute the group key securely
- How to revoke membership of leaving members
- How to prevent joining members from access to past group communication
- How to refresh the group key periodically
- How to log for information and allow for external auditing

The problem of key management for group communication has been the topic of many studies [110, 32, 163, 37, 151, 160, 78, 74, 127]. Some, if not all, of these proposals suffer from drawbacks. The most common is the lack of scalability in the context of re-keying. One case which requires group re-king is for *membership revocation*. This is an essential security requirement, as an old group member must not be able to participate in any of the group's future activities. Also, a recent group member must not be able to access any of the past group activities. Therefore, re-keying and refreshing group members at every change in group status, by joining or leaving members, is a complex problem, and an efficient and scalable re-keying algorithm is required.

Because of their importance and in order to cover the important aspects of multicast security, we give in the following paragraphs a brief overview of several key distribution protocols. They could be categorized into manual, as in the first scheme; centralized, as in the second; pairwise keying similar to the third one; or based on hierarchical trees, as those in the last two protocols.

Manual Key Distribution

In this scheme, the symmetric key is delivered without the use of public key exchange. The root of multicast session would distribute the group key and spare group key to group participants through centralized physical intermediate locations. At predetermined time or in case the group key has compromised, the spare group key is used instead. Manual keying is not appropriate for dynamic sessions in which membership is not defined prior to the start of the session. However, this scheme is useful in some military environments with a well-structured key distribution architecture.

Key Distribution Centers (KDC)

There are several electronic mechanisms for generating and distributing symmetric keys to several computers (for example, communication groups). These techniques generally rely on a Key Distribution Center (KDC) to act as a group coordinator, between the group members, in setting up the symmetric group-key. Military systems such as BLACKER, STU- II/BELLFIELD, and EKMS, and commercial systems such as X9.17 and Kerberos, all operate using dedicated KDCs. A group key request is sent to the KDC via various means (on- or off-line). The KDC acting as an access controller decides whether or not the request is proper, that is, all members of a group are cleared to receive all the data on a group). The KDC would then call up each individual member of the group and download the symmetric key. When each member has the key, the KDC would notify the requester, and then secure group communication could begin. While this was certainly faster than manual techniques, it still requires considerable of set-up time. Also, a third party whose primary interest is not the communication needs to become involved. KDC does not scale for wide-area multicasting where group members may be widely distributed across the internet-work, and a wide-area group may be densely populated.

Group Key Management Protocol (GKMP)

The KDC protocol suffers from several problems. One essential problem is the setup latency time especially when the site becomes overloaded. Because KDC provides access control to every group, the workload is concentrated on one site and is subject to a single point of failure, as well as being subject to performance bottleneck. Because KDC also keeps valuable information, it is vulnerable to compromise, which makes it attractive target for potential attackers.

Unlike KDC protocols, which depend on a central key distribution authority to create and distribute encryption keys to participants, Group Key Management Protocol (GKMP) provides the ability to create and distribute keys within arbitrary-sized groups without the intervention of a global/centralized key manager [79]. The GKMP combines techniques developed for creation of pairwise keys with techniques used to distribute keys from a KDC (i.e., symmetric encryption of keys) to distribute a symmetric key to a group of hosts. GKMP delegates the access control, key generation, and distribution functions to the communicating entities themselves rather than relying on a third party (KDC) for these functions. GKMP works for sender initiated multicast protocols and receiver-initiated protocols.

For the sender-initiated multicast protocols, the GKMP operation starts by the creation of a the Group Key. The GKM application, operating on behalf of the originator, selects one member of the group, contacts it, and creates a group key packet (GKP). A GKP contains the current group traffic encrypting key (GTEK) and future group key encrypting key (GKEK). The GKM application then identifies itself as the group key controller, which the member validates, under cover of the GTEK. After creation of the GKP, the group controller distributes the Group Key by contacting each member of the group, creating a session key package (SKP), validating their permissions (check member's certificate against group parameters), and creating a group rekey. When the group needs to be rekeyed, the originating GKM application selects a member, creates a new GKP, creates a new GRP (which is encrypted in the previously distributed next GKEK) and broadcasts it to the group.

For the receiver-initiated multicast protocols, the GKMP operation starts by the selection of Group Key Controller, a group member who will be made responsible for initial group establishment and periodic generation and dissemination of new GRPs. There is no need for the selected controller to be the controller for all time, but at any one time only one controller may be active for each group. Selection of controller may be made through a voting system, by a simple default (the first to transmit to the group is the controller), or configuration. The group rekey and group redistribution is essentially identical to the sender-initiated multicast protocol.

In general, GKMP is a complex protocol, but one of the essential concepts behind it, is the delegation of group control to the group members. This avoids potential single points of failure, communication congestion, and processor overloading.

Core Based Tree (CBT) Key Distribution

Network Layer multicast protocols such as DVMRP and MOSPF do not have their own protocol header and so cannot provide security in themselves. They must rely on whatever security is provided by IP. On the other hand, CBT multicast protocol makes explicit provision for security: it has its own protocol header, which eliminates the need for other security protocols such as IPSec to provide security services. A Scalable Multicast Key Distribution (SMKD) scheme is built based on this architecture. CBT is based on hard-state type of routers, that is, routers on the delivery tree know the identity of their tree neighbors. The CBT architecture [13, 11] not only provides a solution to the secure joining to the multicast tree, but also provides a solution to the multicast key distribution. The key distribution algorithm can take the advantage of the hard state by appending security information into hard state of the tree, such as access control list, the group key, and the key encryption key. The algorithm contains the following: (1) the initiating host first communicates, via asymmetric encryption, to the Access Control List to a core router, which acts initially as GKDC; (2) the core router generates the group key and the key encryption key; (3) when a new non-core router joins to become a part of the multicast tree, the core router authenticates the new non-core router and "passes" some functionality of GKDC to the non-core router using asymmetric encryption; and (4) the multicast tree expands, the authenticated non-core router further authenticates other new incoming non-core routers, and distributes the security information such as group key and access control list. This approach is highly scalable since each group has a group key distribution center, and is thus regarded as a true distributed KDC. However, it is tied to a specific routing protocol, and does not provide a separation between routing and security. One main assumption is made to put trust on the routers, since each router in the delivery tree obtains the same keys as the group initiated.

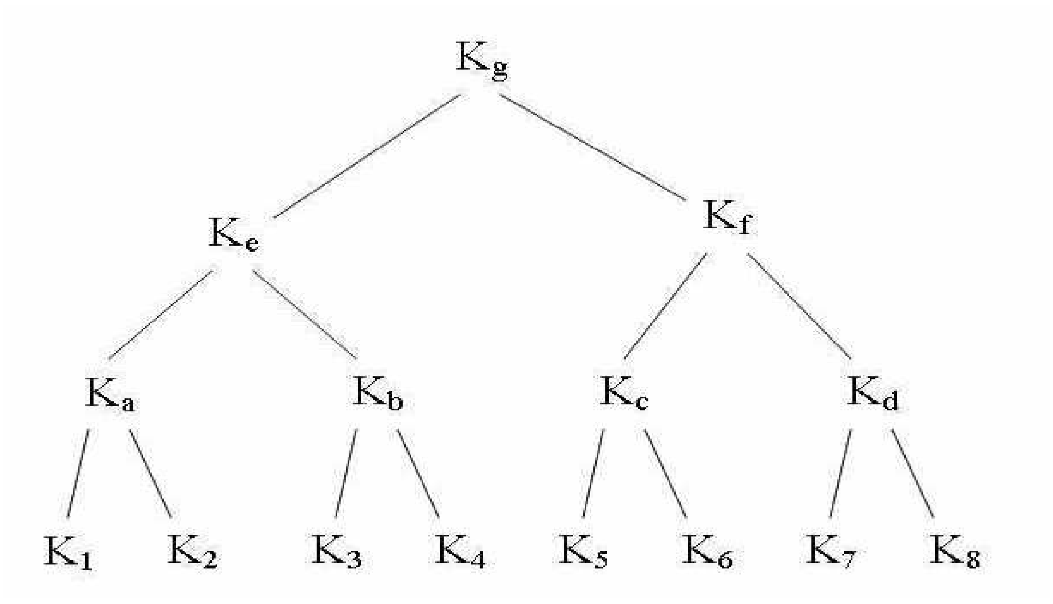

Figure 3.1: Hierarchical Tree key distribution

Hierarchical Tree key distribution

The Hierarchical Tree Key distribution of Wallner *et al* [160] is an efficient and scalable approach that supports dynamic group membership. It focuses on initializing the multicast group with a common net key and rekeying the multicast group. It also identifies techniques, which allows for secure compromise recovery, while being robust against collusion of excluded users. Those features have not been addressed in other multicast key distribution proposals. The objective of this scheme includes minimizing the time required for setup, storage space for each end user and total number of transmissions required for setup, rekey and maintenance. In this scheme, the nodes are arranged in a tree-shaped structure which is basically a root, intermediate nodes, and the leaf. Each element of the structure holds a key supported by the key server of the group. Each leaf holds a pairwise key established between the server and the member, each intermediate node also holds a key generated by the server, and finally the root holds the group key. The key server sends to each member (leaf) a sequence of keys consisting of all keys on the path from the root to the leaf. To ensure security, each key is encrypted with the previous key in the sequence.

For example, in Figure 3.1, user 3 receives intermediate keys K_b, K_e, K_g, where K_b is encrypted with K_3, K_e is encrypted with K_b and K_g is encrypted with K_e. If user 3 is to be expelled, then it requires the key server to generate the keys K_b, K_e, K_g. Each new key is encrypted using its two immediate child leaves. In our case, K_b is encrypted with K_4, K_e is encrypted with both key K_a and K_b, and K_g is encrypted with both child keys K_e and K_f. Thus, given a group of size N, each membership update rekey requires $log(N)$ number of key exchanges.

Although this scheme is optimal in terms of number of rekeying operations, Chu *et al* [37] define two problems with this scheme. First, there is a security loophole where the key server sends a rekey message but the senders it do not receive on time. As a result, the sender continues to encrypt the message with the old group key. In the meantime, a recent expelled member is still able to decrypt multicast data using the old group key, which is considered a security violation. The second problem of this technique is the failure of the algorithm in the presence of a long network delay or lost messages.

3.3.3 Source authentication

The other distinguishing security problem in multicast is source authentication. Authentication services provide assurance to a participating party of the identity and the data of the sender. It guards messages against impersonation, substitution or spoofing. Within the authentication process, a group member is able to verify that the group communication originates from a source within a group; this is called *group authentication.* The process by which group members are able to verify the identity of the sender of the data to the group members is called *source authentication.* Authentication is also an essential part of any key distribution protocol. Applying an authentication mechanism to transmitted multicast group data provides a strong level of integrity.

Loosely speaking, *a scheme for message authentication* should satisfy the following:

- each of the communicating parties can *efficiently produce an authentication tag* to any message of his choice;
- each of the communicating parties can *efficiently verify* whether a given string is an authentication tag of a given message; but
- *it is infeasible for an external adversary* (i.e., party other than the communicating parties) to produce authentication tags to messages not sent by the communicating parties.

An ideal broadcast authentication protocol should be efficient for the sender and the receiver, have a small communication overhead, allow the receiver to authenticate each individual packet, provide perfect robustness to packet loss, and scale to large numbers of receivers. Some other issues which are related to authentication are ensuring *non-repudiation* and providing *anonymity* services (that is, to hide the identity of the group member). These requirements make an efficient authentication scheme in multicast much harder.

Our focus in this thesis is on source authentication. In multicast, we can identify four stages in which source authentication should be performed:

Joining mode

Currently, group membership management in IP multicast and anycast can be used in order to launch denial-of-service attacks. The root of the problem is that routers cannot determine if a given host is authorized to join a group. Therefore, some sort of authentication protocol is needed to restrict the access to group members. The authentication process should start from the beginning, when a host joins a multicast group. This takes the form of one-to-one relationship between the host and multicast group administrator or router. An important condition is that only legitimate users are able to join the multicast group.
Their are two optional solutions for this problem:

- authentication with symmetric-key
- authentication with digital signature

With symmetric-key authentication mechanisms, there is no knowledge difference between senders and receivers since they both use the same key. In other words, an entity that is able to check the validity of a received message is also able to create valid messages itself. This means that all participating IGMP entities must trust each other. An IGMP host that receives a query message with a valid authentication value cannot know, for example, whether this query really came from the querier or from another host impersonating the querier.

With digital signature, it is possible to check the identity of the sender, and it is always possible to check if the receiver tries to repudiate the original message to a third party.

One-to-many mode

At this stage, a group member may need to send a message to the rest of the group members. A recipient may need to verify if this message was sent from a legitimate group member. This takes the form of one-to-many mode.

A typical scenario of this sort of multicast communication is radio or TV broadcasting. Most of the research on source authentication has concentrated on this mode. For example, authenticating streaming applications are important and challenging problem. The problem of continuous stream authentication is solved for the case of one sender and once receiver via standard mechanism such as IPSec. The sender and receiver agree on a secret key which is used in conjunction with a message authentication code (MAC) to ensure authenticity of each packet of the stream.

In the case of of multiple receivers, however, the problem become much harder to solve, because the symmetric-key approach would allow any one of the receivers holding a key to forge the packets or claim it be the original sender. Alternatively, the sender can use digital signatures to sign every packet with its private key. This solution provides adequate authentication, but digital signatures are prohibitively inefficient.

Real-time data streams are lossy, which makes the security problem even harder. With many receivers, we typically have a high variance among the bandwidth of the receivers, with high packet loss for the receivers with relatively low bandwidth. Moreover, authenticity should take place even in the presence of packet loss.

The main features of an efficient broadcast authentication protocol are the following [121]:

- *Efficient generation and verification.* The generation and verification overhead for authentication information should be small. It is important that the verification overhead is small, since a large number of receivers need to verify the authentication information, and some receivers might have restricted computation power.

- *Instant authentication.* Many applications such as stock quote broadcasts, are delay sensitive, and require real-time data authentication. Thus, no packet buffeting is required at signing or at verification.

- *Individual message authentication.* The receiver can authenticate each individual message it receives.

- *Robustness to packet loss.* Internet broadcasting is lossy in nature and many application lost packets are not retransmitted. Hence, the authentication protocol should tolerate packet loss.

- *small size overhead.* Since each message is authenticated independently and instantly, each message carries authentication information. The ideal scheme is the once which holds the least size overhead.
- *Scalability.* Broadcast application have a potentially large number of receivers. The authentication information should be independent of the number of receivers.

Many-to-one mode

Source authentication in multicast is not restricted to one-to-many communication. Report-in style applications, polling data collection, auctions, and juke box usually send their acknowledgment back to the sender in many-to-one mode. The original sender (the receiver and verifier now) needs to check the authenticity of these messages. The main problem which may face the verifier is the *implosion* of messages that need to be authenticated. The receivers usually send back their acknowledgments almost simultaneously; therefore, verification should be extremely efficient and fast to process the large number of messages.

Intermediate nodes

In any communication network, authenticating a flow at transit nodes is preferred. This is recommended to stop special sorts of attacks such as denial of service in their early stages before they are propagated to the rest of the network nodes.

Multicast-related modes

Anycast address is a group address that is assigned to more than one interface. As opposed to multicast, a packet sent to the address would not be routed to all members of the group, but to the source's nearest one only. A host that wants to join an anycast group will have to use a group membership protocol.

Chapter 4

Authentication of Multicast Streams

We first classify the stream authentication problem in the multicast environment and group them into signing and MAC approaches. A new approach for authenticating digital streams using Threshold Techniques is introduced. The main advantages of the new approach are in tolerating packet loss, up to a threshold number, and having a minimum space overhead. It is most suitable for multicast applications running over lossy, unreliable communication channels while retaining the security requirement. We use linear equations based on Lagrange polynomial interpolation and Combinatorial Design methods.

4.1 Introduction

Communication in a computer network may be established as a unicast or multicast connection. In a unicast connection, messages are flowing from a single sender to a single recipient. In a multicast connection, however, a single sender transmits messages to many receivers. Distribution of pay-TV channels is typically done using multicast communication. Protection of multicast messages normally includes both their confidentiality and authenticity. However, in the majority of network services, confidentiality does not seem to be the main concern. Authentication of messages is normally the main security goal. This is true in the Internet environment where most web pages are publicly available while their authentication is the major security worry: the lack of proper authentication is always an attractive target for hackers with an inclination for practical jokes.

Source authentication is the main theme of this chapter. This problem has already been addressed for unicast connections. For instance, the IPSec [90, 88, 89] protocol suite or IP version 6 on the network layer supports source authentication. Higher layers on the OSI reference model make use of authentication services provided by Secure Socket Layer (SSL). Authentication is normally based on message authentication codes

(MACs) generated using private-key cryptography, where the sender and receiver share the same secret key. This approach cannot be easily extended to cover authentication for multicast connections. Public-key cryptography offers an alternative solution for authentication which is ideal for multicast communication; the transmitted message is digitally signed by the sender (holder of the secret key) and everybody can verify the validity of the signed message by using the matching public key. Unfortunately, both generation and verification of digital signatures are very expensive as they consume a large amount of computing resources. Additionally, digital signatures are excessively long, normally more than 778-bits, consuming extra channel bandwidth.

Multicast is commonly used for many real-time applications such as multimedia communication for distant learning and dissemination of digital video. Some distinctive features of multicasting include:

- possible packet loss at some (or perhaps all) destinations, here there is no standard reliable multicast IP protocol yet
- the application tolerates some packet loss but does not tolerate an excessive delay.

Authentication of multicast messages (streams) has to

- provide strong security: no outsider is able to insert packets with forged contents; all packets accepted at destinations have to be genuine
- tolerate loss of packets: any outsider or/and noisy channel may erase a strictly defined number of packets. The number is typically determined by the application. The application refuses to work correctly if the number of accessible packets drops below the limit
- be fast and efficient

This chapter is structured as follows. Section 4.2 reviews related work. Section 4.3 describes the model and states the assumptions for a multicast environment in which the proposed solutions are to be based. In Section 4.4, we present two schemes using linear equations; and in Section 4.5 we present two schemes based on Combinatorial Designs.

4.2 Related Work

The naive solution to authenticate a digital stream is to sign each packet in the stream individually. This way of authentication does not allow detection of the change of order, loss, duplication, and so on. The receiver checks the signatures of packets as they arrive and stops processing the stream immediately if an invalid signature is discovered. Although immediate authentication is possible using this technique, it is extremely inefficient and impractical since it incurs very high computational and space overhead per packet. The problem is worst in broadcast communication.

There are many interesting works that deal with multicast stream authentication (see [64, 31, 164, 122, 133, 67]). The common feature and key idea to most of the above techniques is to amortize a number of authentication operations in one operation, but they differ in their way of handling security or manner of authentication in case of packet loss. The solutions proposed there can be divided into two broad classes. The first includes solutions based on MACs (private-key cryptography). The second employs digital signatures (public-key cryptography).

Gennaro and Rohatchi (GR) [64] used digital signatures to authenticate multicast streams. They considered two cases: on-line and off-line. In the off-line version, they used a chaining of packets. The digest of packet P_i depends on its predecessor P_{i+1}. The first packet is signed using a one-time signature. This technique is efficient and introduces a minimum packet redundancy but does not tolerate packet loss. In the on-line version, each packet is signed using one-time signature and the packet contains the public key that is used to verify the signature of the preceding packet.

Wong and Lam (WL) [64] applied Merkle signature trees to sign a stream of packets. In this scheme, a block of packets are buffered to form a leaf of a tree, in which the message digests of the packets are computed and the root of the tree is signed. The root is the hash of all message digests of the block. Each packet attaches the hashes of other packets in the block as well as the root of the tree to form ancillary information. The number of attached hashes depends on the degree of the tree. They use star, two-level tree, and full binary tree. The scheme is designed to run at sufficient speed for real-time applications, to add a bearable space overhead from authentication data, and to tolerate packet loss gracefully. It was argued in [4] that this scheme increases per-packet computation with packet loss rate, especially in the case of mobile receivers that have smaller computational power and thus have higher packet loss. In their work they tried to overcome this shortcoming by proposing an enhanced scheme.

Rohatchi [133] extended the idea of [64] by computing hash values ahead of time to reduce the time delay for signing a packet. He extended the one-time signature scheme to a k-time signature scheme, and reduced the communication and space overhead of one-time signature scheme. The scheme uses 270 bytes for signature, the server has to compute 350 off-line hash values, and the client needs 184 hash values to verify the signature.

MACs were used in [31] and [122]. As we mentioned earlier, using MACs for multicasting needs special care. The scheme of Canettie *et al* uses asymmetric MAC. The idea behind this technique is that the sender holds n keys and shares half of these keys with each receiver of a group such that no two receivers hold the same set of keys that are held by another receiver. The keys are distributed in such a way that it guarantees that no receivers up to W could collide to reveal the n keys held by the sender; otherwise it would violate the security condition of the scheme. The sender sends a packet attached with n number of MAC's per packet, while the receiver verifies only the MAC's in the packet that holds its corresponding keys, and ignores the rest. This solution does not suffer from the packet loss problem, and each packet is authenticated individually. However, it suffers from the space overhead incurred from extra digest messages attached to each packet.

Another proposal based on MACs is used in Tesla [122]. It uses MACs with delayed key disclosure sent by the sender using a chaining technique on a periodic basis. The idea is to let the sender attach to each packet a MAC, which is computed using a key k known only to it. The receiver buffers the received packet without being able to authenticate it. In time period d, which is set in advance, the sender discloses the key k that is used to authenticate the buffered packet on the receiver side. Packets that are received after the time period are discarded. Actually, the key k is used to authenticate all the packets in the period interval d. The sender and receiver have to synchronize their timing to ensure correct key disclosures.

Golle and Modadugu (GM) in [67] exploit the idea in [64] and introduce a scheme which resists a random packet loss rather than worst-case packet-loss. They prove that their scheme is resistant to burst packet loss given the resources available to the sender and the receiver, and that it has the lowest communication overhead among other the proposals.

4.3 The Model

Messages we would like to authenticate come from a single source and are distributed to multiple destinations (multicasting). Authentication by signing the whole message is not an option when the message is very long or, alternatively, if the transmission of the message may take an unspecified period of time (the length of message is not known beforehand). Real-time live TV communication is of this type. On the other hand, a vendor who offers a video channel normally knows the length of the transmitted message, so the verification of message authenticity could be done at the very end of the movie, but it can fail with a very high probability in case of some lost or/and existence of corrupted packets (multicast connections are not reliable).

Given a long message M (also called a stream), the message is divided into blocks of well-defined size, each block consisting of n datagrams (packets). Assume that the stream is authenticated by signing blocks. There are two possibilities:

1. blocks in the stream are signed independently; attackers may change the order of blocks without receivers noticing it. To prevent this, blocks may contain the sequence number or other time-stamped information. This possibility is suitable for applications that tolerate loss of blocks. For instance, for live TV, loss of a block is typically perceived as a momentary freeze of the frame;

2. blocks are signed in a chaining mode; the signature of the current block depends on its contents and on all its predecessors. Attackers are no longer able to manipulate the order of blocks. Loss of a block, however, makes authentication of all blocks following the lost block impossible. This option suits applications that do not tolerate loss of blocks.

Consider a block B which consists of n datagrams so $\mathrm{B} = (m_1, \ldots, m_n)$. Our attention focuses on authentication of a single block so that:

1. verification of authenticity of blocks is always successful if receivers obtain at least t datagrams (out of n sent); authentication tolerates loss of up to $d = n - t$ packets,

2. expansion of packet contents is minimal; redundancy introduced to the block is minimal under the assumption that authentication tolerates loss of d datagrams;

3. generation of signatures and their verification is efficient, that is, computing resources consumed by them is reasonable (in other words, the delay caused by them is tolerable by the applications run by receivers).

The last point needs further elaboration. The sender (source) is normally much more powerful than recipients. If this is true, then authentication can be performed as follows:

- a powerful sender generates (expensive) digital signatures independently for each block;
- a receiver applies batch verification of signatures, that is, merges signatures of blocks and the result can be verified as a single signature.

This solution is quite attractive, especially for applications which tolerate longer delays, in which the validity of blocks can be asserted only after receiving the last signature in the batch. However, this solution must be applied with extreme caution as attackers may inject invalid signatures and false messages which will cancel each other in the batch (the verification process gives OK when, in fact, false blocks have been injected to the stream).

More formally, the stream authentication scheme is a collection of three algorithms:

- $(S_k, P_k) = \texttt{GEN}(1^v)$: is a key generation algorithm that generates the secret-key S_k and the corresponding public-key P_k, based on some security parameter v,
- $\mathrm{A} = \texttt{AUTHEN}(\mathrm{B}, S_k)$: generates authenticator A for a block B with the aid of the sender secret key S_k,
- $\texttt{Ver}(\mathrm{B}, \mathrm{A}, S_k)$: verifies authenticator A which is attached to the block B using sender's public key P_k. The result is binary and can be either OK or FAIL.

The scheme must :

- *tolerate loss of packets* within the block: any t out of n packets allows the running of the algorithm `Ver`, which can generate either OK or FAIL
- be *secure*: any malicious/accidental change of either contents or order of packets within the block must be detected by receivers,
- introduce *minimal redundancy* to the block B, or in other words, the bandwidth expansion should be minimal
- be *efficient*: execution of both algorithms `AUTHEN` and `VER` should consume a small amount of computing resources (if the sender is powerful, then efficiency of `AUTHEN` is not crucial). The above requirements are typically referred to as the design goals.

4.4 Stream Authentication Based on Linear Equations

We start from the following authentication scheme, which authenticates a single block B using a digital signature generated for the hash value of the block B. The verification algorithm can be run if all hash values of packets are available. In our scheme, t hash values are obtained directly from packets (we assume that t packets from the block have arrived safely at the destination). The missing d hash values are computed from the information attached to an extra packet m_{n+1}. Specifically,

- n: is the total number of packets to be sent in a block B.
- t: is the threshold number of packets needed to arrive safely at the destination in order to be able to authenticate the whole block B
- d: is the number of packets in the block B that is tolerated to loss.

Therefore, the relation between the above parameters could be formulated as:

$$n = t + d, \quad d \leq t \leq n.$$

4.4.1 Scheme 1

```
AUTHEN(B, $S_k$):

  1. Divide the message block B into $n$ datagrams $m_1, \dots, m_n$

  2. Create $h_i = H(m_i)$ for $i$ =1, ..., $n$ where $H$ is a collision resistant
     hash function such as SHA-1

  3. Form a polynomial $F(x) = \sum_{i=1}^{n} h_i x^{i-1}$
     and compute $d$ control values $F(j)$ for $j = 1, \dots, d$, where $d \leq t$

  4. Compute the message digest $D = H(h_1, \dots, h_n)$

  5. Sign the computed value $D$ using a strong cryptographic signature $Sig_k$

  6. Return datagram:

         $m_{n+1} = F(1)\|, \dots, \|F(d)\|Sig_k(D, S_k)$,

     where
         $Sig_k(D, S_k)$
     is the digital signature generated in step 5.

Ver ($\tilde{B}, m_{n+1}, P_k$):

  1. The block must contain at least $t$ datagrams, where $t \leq n$.
     If this does not hold or $m_{n+1}$ does not exist, the algorithm returns FAIL

  2. If the number of packets in the block is at least $t$, then order the datagrams
     correctly so that the sequence is identical with the one in the source
     with at most $d$ missing datagrams

  3. Find all the hash values $h_i$ for the packets from $\tilde{B}$.
     Note that at least $t$ hash values are generated.

  4. Compute the missing hash values for lost datagrams from the $d$ control values
     $F(j)$, where $j = 1, \dots, d$. In other words, solve the system of linear equations
     generated from the control values.

  5. Verify the signature using a native (to signature) verification algorithm and the
     public key of the sender. If the signature is correct, return OK; otherwise, exit
     FAIL.
```

Consider the design goals. The first goal, *tolerance for packet loss*, is not achieved. The scheme works only if the $(n+1)$-th packet containing control values arrives at the destination. Consider the case when $t \leq n$ packets arrive safely at the destination together with the last packet m_{n+1}. Without loss of generality, assume that these packets are $m_1 \ldots m_t$. The receiver can run the verification algorithm `VER` if it is able to compute the digest D of the whole block. To do this, it must be able to recover the hash values of all packets. First, they compute values directly from the available packets. Next, they compute the missing ones from the control values accessible in the $(n+1)$-th packet. In other words, they solve the following system of linear equations:

$$\begin{aligned} F(1) - a_1 &= h_{t+1} + \ldots + h_n \\ F(2) - a_2 &= h_{t+1} \times 2^t + \ldots + h_n \times 2^{n-1} \\ &\vdots \\ F(d) - a_d &= h_{t+1} \times d^t + \ldots + h_n \times d^{n-1} \end{aligned}$$

where

$$a_r = h_1 + h_2 r + \ldots + h_t r^{t-1}$$

for $r = 1, \ldots, d.$

The above system of linear equations always has a solution as that characterized by a Vandermonde matrix that is always nonsingular (whose determinant is nonzero). Having all hash values, the receiver computes the digest D of the block B and verifies the signature (using the sender's public key).

We believe that the security of the scheme is equivalent to the security of the underlying digital signature (assuming that the hash function is collision resistant). To see this, it is enough to note that the only difference between the underlying signature and the signature used in the stream authentication scheme is the way the verification is being performed. For stream authentication, the verifier computes at least t hash values from packets that safely arrived at the destination, while d (or fewer) missing ones are computed from the control values. For the original signature scheme, the verifier computes all hash values directly from messages.

Data expansion is indeed minimal as the receivers get the minimum necessary information to successfully run the verification algorithm. This statement is true on the assumption that hash values are much smaller than the payload of datagrams. In practice, the length of a hash value is 128 bits (for MD5)[1] or 160 bits (for SHA-1) while the maximum length of packets in X-25 is 8192 bits.

4.4.2 Scheme 2

As noted, the scheme considered above is not tolerant of the loss of packets. More specifically, the absence of the $(n+1)$-th packet precludes the receiver from getting the system of linear equations, which are necessary to generate the digest D of the block B.

The algorithm in the next pages presents the second scheme, which does tolerate a loss of d packets. The main idea is to distribute the content of the $(n+1)$-th packet among other packets using a system of linear equations (in a fashion very similar to what has been done for hash values).

The only aspect of this scheme which needs some clarification is how the receiver reconstructs the redundant message R, or equivalently finds the polynomial $R(x)$. Note that the receiver knows at least t points lying on the polynomial $R(x)$. As the polynomial is of the degree $(t-1)$, any t different points uniquely determine a polynomial, which contains that points. This can be done very efficiently using the Lagrange interpolation.

[1]MD5 is a no more secure message digest algorithm

<u>`AUTHEN`(B, S_k) :</u>

1. Divide the message block B into n pieces:

$$m_1, \ldots, m_n$$

2. Compute:

$$h_i = H(m_i)$$

for $i = 1, \ldots, n$ and the block digest

$$D = H(h_1, \ldots, h_n)$$

where H is a collision resistant hash function.

3. Create a polynomial:

$$F(x) = \sum_{i=1}^{n} h_i x^{i-1}$$

4. Form a redundant message:

$$R = (F(1) \| F(2) \|, \ldots, \| F(d) \| Sig_k(D, S_k))$$

and split it into t pieces of the same size, i.e.

$$R = b_0, \ldots, b_{t-1}$$

where Sig_k is a strong cryptographic signature s.

5. Create a polynomial:

$$R(x) = b_0 + b_1 x + \ldots + b_{t-1} x^{t-1}$$

and define n control values $R(1), \ldots, R(n)$ that are assigned to respective datagrams.

6. Return the sequence of datagrams each of the form for $i = 1, \ldots, n$.

$\texttt{Ver}(\tilde{B}, P_k)$:

1. The block $\tilde{B}$ must contain at least t datagrams. If this does not hold, the algorithm returns FAIL.
2. If the number of packets in $\tilde{B}$ is at least t, then order the datagrams correctly so the that sequence is identical with the one in the source with at most d missing datagrams.
3. From each datagram extract the contents and the corresponding control value $R(i)$.
4. Find all the hash values h_i for the packets from $\tilde{B}$. Note that at least t hash values are generated directly from m_i.
5. Assemble a system of linear equations from the available control values $R(i)$, solve it and determine the redundant message:

$$R = (b_0, \ldots, b_{t-1})$$

6. Extract the control values $F(i_j)$ for the missing j and reconstruct the hash values of missing datagrams.
7. Verify the signature using a native (to signature) verification algorithm and the public key of the sender. If the signature is correct return OK; otherwise, exit FAIL.

4.5 Stream Authentication Based on Combinatorial Designs

In this section, combinatorial design methods will be used to distribute message digests in each of the packets in such a way that *all* message digests are retrieved when at most d packets are lost. Two methods are represented here.

4.5.1 Balanced Incomplete Block Design

The arrangement of the message digests into the packets uses the combinatorial technique of Balanced Incomplete Block Design (BIBD) [71]. A BIBD (v, b, k, r, λ) is an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, and each object occurs in exactly r different blocks. In BIBD, every (unordered) pair of distinct objects occurs together in exactly λ blocks. In our case, the objects are the message digests, and the blocks are the packets. The problem is to distribute the v message digests into the b packets in such a way that each message digest appears in r packets, and each packet holds k different message digests. By the very definition of *BIBD*, if up to $d = r - 1$ of such packets are lost, we can still retrieve all the k message digests from the remaining $t = v - d$ received packets.

As an example, suppose we have $v = 7$ message digests that need to be distributed over $b = 7$ packets. Then, a BIBD (7, 7, 3, 3, 1) will allow us to retrieve all the 7 message digests from any $t = 5$ of the received packets (i.e. 30% packet loss), as the following listing of the packets shows (the message digests are coded to integers $0 \ldots 6$):

$$P_0 : \ 0, \ 1, \ 3$$

$$P_1 : \ 1, \ 2, \ 4$$

$$P_2 : \ 2, \ 3, \ 5$$

$$P_3 : \ 3, \ 4, \ 6$$

$$P_4 : \ 4, \ 5, \ 0$$

$$P_5 : \ 5, \ 6, \ 1$$

$$P_6 : \ 6, \ 0, \ 2$$

This special case when $v = b$ is called the Symmetric Balanced Incomplete Block Design (SBIBD). The necessary condition for this sort of design is $\lambda(v - 1) = k(k - 1)$. Since in our applications, the number of message digests is always equal to the number of packets, and as each packet has a corresponding message digest, we will always exclusively deal with SBIBD in these applications. Note that it is possible to further reduce the number of message digests in the above example in each packet to 2 messages, since it is possible to compute the message digest of the holding packet directly from the packet itself. The enhancement is as follows:

$$
\begin{aligned}
P_0 &: \ 1,\ 3 \\
P_1 &: \ 2,\ 4 \\
P_2 &: \ 3,\ 5 \\
P_3 &: \ 4,\ 6 \\
P_4 &: \ 5,\ 0 \\
P_5 &: \ 6,\ 1 \\
P_6 &: \ 0,\ 2
\end{aligned}
$$

There are many ways of constructing SBIBD designs that are found in the literature and for different choices of parameters. Our choice was focused on the design that yields the highest ratio of k to v, that is, the minimum number of message digests that needs to be carried by each packet, relative to the total number of packets that need to be present to authenticate a block. In other words, we are looking for a high ratio of packet loss while still able to authenticate the block. Colbourn and Dinitz [38] list over 12 families of SBIBD(v, k, λ) designs for various values of parameters. One of these families refers to Hadamard designs, and they correspond to designs with parameters $v = 2^n - 1$, $k = 2^{n-1} - 1$ and $\lambda = n - 1$. A Hadamard *SBIBD* design exists if and only if $H(4n)$ exists, where H is a Hadamard matrix of order $n \times n$. The ratio of k to v is as high as approximately half, which means it is possible to authenticate the block in the presence of half of the packets of the block (i.e. 50% packet loss).

To construct a Hadamard matrix for a group of 2^n packets, the matrix is normalized by suitable row and column negations. Removing the first row and column leaves a $2^{n-1} \times 2^{n-1}$ matrix, the core of the Hadamard matrix. Replace all -1 entries with 0 entries, the result is an SBIBD$(2^{n-1}, 2^{n-1}, n-1)$. The 1 entries actually represents the arrangements of the v entries (message digests) over b (packets). These matrices designs could be prepared according to the proper sizes and selected during the implementation.

To construct such a method, a number of packets of size 2^n are grouped to form a block. Then, a Hadamard matrix of order $2^n \times 2^n$ is constructed. Finally, message digests are assigned to the corresponding packet index in the block according to the Hadamard matrix [58].

One advantage of the SBIBD approach is that there is no additional computation overhead cost involved in computing the threshold parameters and retrieving them by sender and receiver. The additional space overhead is minimum compared to other techniques as in [31, 164].

4.5.2 Rotational Designs

Another dynamic, simple and easy way in which it is possible to control the degree of the threshold is by using Rotational Designs. The example in Table 4.1 demonstrates the distribution of 6 message digests over a block of 7 packets.

Packet Number	*Attached Message Digests*					
0	1	2	3	4	5	6
1	2	3	4	5	6	0
2	3	4	5	6	0	1
3	4	5	6	0	1	2
4	5	6	0	1	2	3
5	6	0	1	2	3	4
6	0	1	2	3	4	5

Table 4.1: Distribution of message digests using SBIBD

In this distribution, $v = b = 7$ and $k = r = 6$. The column represents the packet number and message digests of packets it holds, and the rows represent the message digests distribution over packets. Now, if we want to apply the threshold scheme and require only one space overhead per packet but with high availability (i.e. one packet loss maximum), then the first row only of table 4.1 will be distributed. If, however, two message digests are attached to each packet, then a maximum of two packets are tolerated to loss. Therefore, the first and the second row distribution will be applied only and so on. The distribution in Table 4.1 is the worst case, and typically used in the Wong and Lam scheme [164]. Formally speaking, each packet holds the message digests of the rest of the packets in the block, and consequently each block is authenticated individually. Obviously, their solution suffers from space-overhead (large packet size) resulting from appending the message digests to the original packet, and this is even worse when counting the block signature space.

4.6 Comparative Analysis

In this section, we evaluate the efficiency of our two methods of stream authentication protocols by setting a comparison of the performance against other known or standard algorithms.

4.6.1 Linear Equations method

It was argued at the beginning of this chapter that using typical digital signature for authenticating digital stream by signing each packet in a block of stream is an expensive operation. Rather, we proposed using linear equations. To illustrate this, we compare the complexity of our proposed protocol in Section 4.4 against the complexity of the Digital Signature Algorithm (DSA). We choose ElGamal [57] digital signature as one of the algorithms which belongs in this category.

ElGamal signature generation for a generator α, a secret key k, and random prime p is relatively fast, requiring one modular exponentiation ($\alpha^k \ mod \ p$), the extended Euclidean algorithm (for computing $k^{-1} \ mod \ (p-1)$), and two modular multiplications. Modular subtraction is negligible when compared with modular multiplication. The exponentiation and application of the extended Euclidean algorithm can be done off-line, in which case signature generation (in instances where precomputation is possible) requires only two (on-line) modular multiplications [107].

ElGamal signature verification is more costly, requiring three exponentiations. Each exponentiation (using naive techniques) requires $\frac{3}{2}\lceil log \ p\rceil$ modular multiplications, on average, for a total cost of $\frac{9}{2}\lceil log \ p\rceil$ multiplications. Signature verification calculations are all performed modulo p, while signature generation calculations are done modulo p and modulo $(p-1)$ [107]. The total cost of ElGamal signature to sign a single packet for both generation and verification equals 5 modulo exponentiation operations which is equal to $(5 * (9/2) * log(p))$. If a block B contains $n = 512$ packets, then the total cost for signing and verifying the block for $p = 512$ equals $512 \times (5 \times (9/2) \times log(512))$.

Now, the calculation of the complexity of our scheme is derived by computing the cost of generating the control packet which requires d evaluations of polynomial of degree n. This can be computed by Horner's method [29], yielding complexity of $d * n$ multiplications.

The cost of verification can be derived by solving a system of linear equations of the values held in the control packet m_{n+1}. This packet holds all the information needed to recover the authentication information of the d missing packets needed to verify the arriving t packets. Specifically, the cost of solving a system of linear equations using Gauss-Jordan Elimination [29] requires a number of multiplications which equal to

$$\frac{d^3}{2} + d^2 - \frac{5d}{2} \tag{4.1}$$

We also add to the equation 4.1 the cost of signing the value D by ElGamal signature, which has a cost of 5 modulo exponentiations. Also, we add $2(d * n)$ multiplications, which is equivalent to the cost of solving a system of d linear equations with the same number of unknowns.

$$\frac{d^3}{2} + d^2 - \frac{5d}{2} + 2(d * n) + 5((\frac{9}{2})\ log\ p) \tag{4.2}$$

Our goal is to estimate the highest d which makes our scheme better in cost than ElGamal signatures. By running a computer program, for a block B of $n = 512$ packets, and for $p = 512$, we find that our scheme is better than ElGamal if $d \leq 216$, that is, when setting the loss of packets in a block for a maximum 40% of the total size of the block. Otherwise, ElGamal signature would be rather less expensive in total. This also logically means that, when increasing d, then the complexity tends to increase.

4.6.2 Combinatorial Design method

We compare our protocol SBIBD with the proposed protocols found in Section 4.2, namely Wong and Lam (WL), Tesla, Gennaro and Rohatchi (GR), and Gole. Note that the scheme proposed in (WL) came in three basic flavors depending on the ancillary information which is organized within a group of packets. Note also that Tesla uses a keyed hashing method.

To establish a common platform, consider a stream that is divided into blocks of 16 packets. Assume that all the protocols run a hashing algorithm which produces 20 bytes as a message digest (regardless of keying or non-keying algorithms). Assume a single digital signature is generated by the sender and verified by the receiver for each block. The following table summaries the comparison:

Scheme	Signature	Hash	Overhead	Loss	Delay
WL star	1	17	340	any	0
WL tree	1	21	160	any	0
Gole	1	16	43	burst	16
Tesla	1	17	40	any	0
GR	1	16	20	no	0
SBIBD	1	16	120	threshold	7

Table 4.2: A Comparison of multicast authentication schemes

Legend: WL: Wong & Lam, GM: Golle & Modadugu, GR: Gennaro & Rohatchi

- Signature: the number of signing operation per block
- Hash: the total number of hashes computed by the sender
- Overhead: the overhead per packet in bytes
- Delay: the delay (in number of packets) on the receiver side before authentication is possible
- Loss: possibility of authentication in loss of packets scenarios

Each of the above protocols has a side effect. For example, although Tesla is efficient and versatile, its main drawback is that it requires time synchronizing between the sender and receivers (with some margin). Keys are disclosed in a time interval after sending the packet to the receiver. As a prerequisite security condition of the protocol, a receiver must obtain a packet before the next packet can be sent from the sender. The basic Tesla protocol does not provide non-repudiation security service since it uses MAC. An improved version of Tesla was introduced also in [122]; this provides non-repudiation using a digital signature.

GR is extremely efficient, but its drawback is that it is not resistant to packet loss. The GM approach is elegant, but it has a delay time equal to the entire block size. Our threshold approach of SBIBD is similar to GM, but it has less delay time, with higher overhead size. SBIBD threshold delay of verification still has less overhead size than the WL approach. Our approach can tolerate packet loss up to a threshold number equal to almost half of the block size.

WL provides an important feature in permitting immediate authentication per packet without buffering a range of packets before authentication could be possible. The penalty is in high space overhead carried in each packet.

SBIBD, as a threshold approach, is an intermediate solution between the WL and GR approaches. It allows average packet overhead and average waiting time (presence only of half of the block packets before start authentication i.e. 50% packet loss). Also, it is flexible because of the ability to control the overhead in advance.

We emphasize on the observation in [31], and draw the following conclusion: in multicast applications, no single security protocol could fit all the scenarios of the diverse applications. For instance, some of the authentication protocol may work more efficiently in special environment than other protocols. For example, the GR approach may be an excellent option in a connection-oriented communication network, where packet loss is rare. The GM approach may be suitable in networks with bursty losses of packets. SBIBD, as an example of the threshold approach, has the main benefit in its dynamic behavior. That is, it can be tuned to, according to the environment of the application. If the packet loss is high, the packet can be turned to hold more ancillary information. On the other hand, in error-free communications or virtual-circuits networks such as ATM networks, which guarantee quality of service packets can be tuned to carry less ancillary information.

The proposals above focus on handling the efficiency rather than security since most of the security of these approaches depends on the underlying digital signature or keyed-hashing. GR, for example assumes an efficient one-time signature. Tesla, on the other hand, uses keyed MAC, and has a special mechanism to handle key disclosing. There are other security concerns. For example, since authentication is block-based, an attacker may manipulate the order of the blocks. Consider a piece of information that needs to be treated as one object with other pieces of information (for example, salary with employee name), but which is found in packets that happen to be on different blocks. Reordering the blocks may delude to produce an authentic block, but in fact the information has not been correctly represented. Hence, sequencing the blocks and ordering is a necessary condition for methods which use blocks for authentication. Also, in the block-based authentication schemes, because a stream is divided evenly or randomly, each packet or block does not have any physical meaning itself. Therefore, the receiver may not get correct, useful information from a single packet or block unless its context (previous and following packet/block) is correctly received. When a packet is lost, its contiguous packets or blocks may not be usable. Wu, Ma and Xu in [165] suggested an object-based streaming authentication solution to solve this problem.

4.7 Summary

Streaming applications such as pay-TV have become very popular in recent years. Using multicast infrastructure is the best platform to transport this sort of application. However, authenticating the source of the multicast is a major concern. The problem has already been addressed in a number of proposals. In this chapter, we first classified these proposals; then, we investigated the problem in an unreliable communication environment. Different schemes based on threshold methods have been proposed. The threshold method provides flexibility in controlling the amount of ancillary information needed to be carried in a packet. The schemes try to avoid the drawbacks of other proposals. In general, they have less overhead space and can tolerate packet loss up to a threshold number. In fact, it is difficult to have a single protocol that can fit the diverse scenarios of multicast. Therefore, the diversity of protocols in multicast is not strange.

Chapter 5

Authentication of Concast Communication

In this chapter we consider the problem of finding an efficient signature verification scheme when the number of signatures is significantly large and the verifier is relatively weak. In particular, we tackle the problem of message authentication in many-to-one communication networks, known as *concast communication.*

The chapter presents three signature screening algorithms for a variant of ElGamal-type digital signatures. The cost for these schemes is n applications of hash functions, $2n$ modular multiplications, and n modular additions plus, the verification of one digital signature, where n is the number of signatures.

The chapter also presents a solution to the open problem of finding a fast screening signature for non-RSA digital signature schemes.

5.1 Introduction

One of the greatest outcomes of the invention of public-key cryptography [56] is the digital signature. It creates a sort of digital encapsulation for the document such that any interference with either its contents or the signature has a very high probability of being detected. Because of this characteristic, the digital signature plays an important role in authentication systems. Since digital signatures are transmitted in public channels, they are subject to a variety of attacks, and therefore the verification of digital signatures is a common practice. A verification of digital signature needs to apply a particular (in general, a publicly-known) algorithm. So, a digital signature scheme is a collection of two algorithms, and it must have the following properties:

1. The signing algorithm $Sig_K : \mathcal{K} \times \mathcal{M} \to \Sigma$ generates a signature $\sigma = Sig_K(M)$, where $M \in \mathcal{M}$ is a message, $K \in \mathcal{K}$ is the secret key of the signer, and Σ is the set of all possible values of the signatures.

2. The signing algorithm executes in polynomial time when the secret key K is

known. For an opponent who does not know the secret key, it should be computationally intractable to forge a signature, that is, to find a valid signature for a given message and for a given signer.

3. The verification algorithm $V_k : k \times \mathcal{M} \times \Sigma \rightarrow \{\text{yes, no}\}$ takes public information $k \in \mathcal{K}$ of the signer, a message $M \in \mathcal{M}$ and a given signature $\sigma \in \Sigma$ of the message M. It returns "yes" if σ is the signature of the message M; otherwise it returns "no".

4. The verification algorithm, in general, is a publicly known (polynomial time) algorithm. Therefore, anyone can use it to check whether a message M matches the signature σ or not.

Several digital signature schemes, for different purposes, have been introduced in the literature of public-key cryptography. In original digital signature schemes (e.g., [132, 57]), both parties of the system (the signer and the verifier) are individuals. The invention of society and group oriented cryptography [52] led to the generation and/or verification of digital signatures by a group of participants rather than individuals (see, for example, [27, 54, 75, 45]). In almost all of these digital signature schemes, the generation/verification of a signature requires the performance of some exponentiation. Since exponentiation is a costly operation, the design of efficient digital signature schemes (from both the generation and verification points of view) has been the subject of investigation by many researchers (see, for example, [61, 138, 55, 40]). The efficiency of the system is of paramount importance when the number of verifications is significantly large (e.g., when a bank issues a large number of electronic coins and the customer wishes to verify the correctness of the coins).

5.1.1 Related Work

In our schemes, verification of a digital signature implies also modular exponentiation. Thus, previous works on improving the performance of modular exponentiation [28, 135] and batch verification of modular exponentiation [60, 15] are highly relevant to this work. Bellare *et al* in [15] introduced new technique for batch verification using modular exponentiation. In their work, a batch instance consists of a sequence $(x_1, y_1), \ldots, (x_n, y_n)$, and the query is whether or not $y_i = g^{x_i}$ for all $i = 1, \ldots n$ (where g is a primitive element in a group). Their proposed algorithms solve this problem with an error probability of $2^{-\ell}$ for a predetermined parameter ℓ.

They have also considered batch verification for the RSA signatures, where the verification relation is modular exponentiation with a common exponent. That is, given a sequence of pairs (M_i, σ_i), one would like to verify the ith signature by checking that $\sigma_i^e = H(M_i) \bmod N$, where $H(.)$ is a hash function and e, N are the RSA public key and modulus respectively. In their solution to this particular case (also called *screening algorithm*) the batch instance (M_i, σ_i), $i = 1, 2, \ldots, n$ passes the test if

$$\left(\prod_{i=1}^{n} \sigma_i\right)^e = \prod_{i=1}^{n} H(M_i) \pmod{N}.$$

It is obvious that the batch instance $(M, x\alpha), (M, x/\alpha)$ is incorrect, but it passes their verification test. However, they have shown that this is not really a problem from a screening perspective, since one wants to make sure that M has been sighted by the legitimate signer, even though the signature is not correct. They have proved that if RSA is one-way, then an adversary cannot produce such a batch instance that was never signed by the signer but still passes their test.

Note that, in their work [15], fast screening algorithms for other signature schemes and several other issues have been left as open problems.

Batch verification is a useful technique for our solution to the concast scenario as it is described in Section 5.2.

5.1.2 Concast Scenario

Multicast is a one-to-many communication mode that has greatly interested the research community in the past few years as a tool for delivering messages to a group of recipients in an efficient way. The main benefit behind deploying multicast is that it minimizes the bandwidth utilization in the already congested network [55, 6].

Multicast communication is usually not a one-way direction communication. A group of recipients, in reliable multicast applications for example, may contact the sender as a feedback acknowledgment. A wide range of many-to-one applications also includes shared-whiteboard, collaborative applications, and report-in style applications. This sort of many-to-one communication is known as *concast communication.*

The well-known *implosion* problem in broadcast communication is addressed here. The problem occurs when a receiver is overwhelmed with messages from different senders and has to process them efficiently. The problem of implosion could be worse if the signature verification is required to authenticate a group of messages. In this case, an efficient authentication scheme is required to alleviate the burden on the receiver.

Therefore, concast communication can be considered, from the security perspective, as a many-signers/one-verifier problem.

In this chapter, we present different schemes that solve this problem. Our first scheme works with the help of a trusted combiner. The second scheme works with no help from a trusted party, but requires interaction between signatories. The third scheme, however, minimizes the interaction between parties in the system.

5.2 The Model

Given a sequence of signatures $(M_1, \sigma_1), \ldots, (M_n, \sigma_n)$, a recipient, with relatively small computing resources accessible to him, wishes to verify these signatures. The naive method is to verify each signature individually and to accept the whole set if all signatures pass the verification algorithm. Obviously, this is a very time consuming task and is not applicable for a recipient with small computing power. An alternative method could be to use the batch verification strategy, in which a randomly selected subset of signatures is verified, and, if that subset passes the verification, then we accept (with some probability) that the whole sequence will pass the verification algorithm. However, this technique might only be acceptable if there is an efficient and trusted entity between the receiver and the senders.

A desirable solution could be if the verifier can perform a signature screening and accept the whole set of signatures if they pass the test. In other words, screening is the task of determining whether the signer has at some point authenticated the text M_i, rather than the task of checking that the particular string σ_i provided is a valid signature of M_i. Note that the screening technique of [15] does not seem to be applicable for RSA based signatures in a concast environment. In Section 5.5, we present a signature screening for a variant of ElGamal [57] type digital signatures.

5.3 Components of the System

This section considers the basic tools which we will use for the implementation of our schemes.

5.3.1 Communication Channel

Each signer and the verifier is connected to a common broadcast medium with the property that messages sent to the channel instantly reach every party connected to it. We assume that the broadcast channel is public, that is, everybody can listen to all information communicated via the channel, but cannot modify it. These assumptions for this model of communication channel may seem somewhat unrealistic (i.e. does not fit the Internet or cellular network). However, the purpose of these assumptions is to focus on the proposed protocol at a high level. It is worth noting that these assumptions can be substituted with standard cryptographic techniques for achieving privacy and authenticity using for example signcryption primitive (e.g., see Section 9.5).

5.3.2 Signature Scheme

We employ a variant of ElGamal-type digital signature, which is a slightly modified version of a signature that has been used in [116]. Let p, q be large primes such that $q|(p-1)$, and let $g \in \mathbb{Z}_p = GF(p)$ be an element of order q. Let $H(.)$ be an appropriate hash function that hashes messages of arbitrary length into an element of $\mathbb{Z}_p$. Also let $x_i \in \mathbb{Z}_q$ be the secret key and $y_i = g^{x_i} \pmod p$ be the public key associated with user u_i. The values p, q, g, y_i, and the hash function $H(.)$, are the common parameters in the network.

Signature Generation:

In order to sign a message $m = H(M) \in \mathbb{Z}_p$, the signer chooses a random k and computes

$$r = mg^{-k} \pmod p \quad (5.1)$$

$$s = k - r'x_i \pmod q \quad (5.2)$$

where $r' = r \pmod q$.

Verification:

The verifier accepts the signature (M, s, r) if the following equation holds true:

$$H(M) = g^s y_i^{r'} r \pmod p \quad (5.3)$$

else, return $\perp$ (fail indication).

5.3.3 An Approach to Digital Multisignature

In society and group oriented cryptography, it is required that a cryptographic transformation be performed by a group of participants rather than an individual. Let $\mathcal{U} = \{u_1, \ldots, u_n\}$ be the set of all users and assume that the group policy requires that a group signature is to be mutually generated by all group members. This is known as a *multisignature scheme*. The group signature on message $m = H(M) \in \mathbb{Z}_q$ can be generated using the following protocol:

Signature Generation:

1. Each u_i chooses a random $k_i \in \mathbb{Z}_p$ and computes $r_i = mg^{-k_i} \pmod p$.
2. After all participants broadcast their r_i, every signatory calculates $r = \prod_{i=1}^{n} r_i \pmod p$.
3. Each u_i $(i = 1, \ldots, n)$ generates his signature as $s_i = k_i - r'x_i \pmod q$, where $r' \equiv r \pmod q$.
4. Each u_i $(i = 1, \ldots, n)$ sends his partial signature (s_i, r_i) of message m to the combiner (through the public channel).
5. Once all partial group signatures are received, the group signature of message m can be generated as (s, r), where
$$s = \sum_{i=1}^{n} s_i \pmod q.$$

Verification:

The verification of the group signature is similar to the verification of an individual signature. Note that the secret key of the group is, in fact, $x = \sum_{i=1}^{n} x_i \pmod q$, and the public key of the group is $y = \prod_{i=1}^{n} y_i \pmod p$. The verifier accepts the signature (M, r, s) if the following equation holds true:
$$m^n = g^s y^{r'} r \pmod p$$

Note that the concast scenario is different from the multisignature scheme in at least two ways:

- In a concast environment, the set of users (signatories) is not fixed.
- In a concast environment, each user may sign a different message.

5.4 Concast Signatures

Before introducing the Concast scheme, let us start with a warm-up solution as a preamble to the idea.

5.4.1 Warm-up Solution

Let users $u_1, \ldots, u_n$ wish to sign the messages $m_1, \ldots, m_n$ respectively, where $m_i = H(M_i)$. Consider the following protocol, which works in almost the same manner as multisignature schemes, although the messages are different.

Signature Generation:

1. Each u_i chooses a random $k_i \in \mathbb{Z}_p$ and computes $r_i = m_i g^{-k_i} \pmod p$.
2. After all participants broadcast their r_i, every signatory calculates $r = \prod_{i=1}^{n} r_i \pmod p$.
3. Each u_i $(i = 1, \ldots, n)$ generates his signature as $s_i = k_i - r' x_i \pmod q$, where $r' \equiv r \pmod q$.
4. Each u_i $(i = 1, \ldots, n)$ sends his signature (M_i, s_i, r_i) through the public channel.

Verification:

1. After receiving n signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$, the verifier computes
$$s = \sum_{i=1}^{n} s_i \pmod q, \qquad \text{and} \qquad m = \prod_{i=1}^{n} H(M_i) \bmod p$$
2. The verification of the combined signature (m, s, r) is the same as in the underlying signature scheme, that is, the signatures are accepted if
$$m = g^s y^{r'} r \pmod p$$
otherwise, return $\perp$.

Performance Issues

Given n signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ the scheme requires n applications of hash functions (to generate $H(M_i), i = 1, \ldots, n$), n modular multiplications (to compute m), n modular multiplications (to compute r), and n modular additions (to generate s) in order to construct the combined signature (m, s, r). After getting the signature (m, s, r), the verifier needs to verify a single signature as in the underlying digital signature scheme. This means, from an efficiency point of view, that the cost of our scheme is n applications of hash functions, $2n$ modular multiplications, and n modular additions, plus the verification of one digital signature.

However, from a practical point of view, the scheme needs some interaction (i.e. message exchange) between the signatories. Although this is a common practice in almost all society-oriented cryptographic systems, it may not be reasonable in a concast environment, since the signatories may not form a group. In the next scheme, we will present a protocol that works with no interaction between the signatories.

5.4.2 The Concast Scheme

We now present a modified version to the previous algorithm in which no interaction between the signatories is required. In this algorithm, instead of broadcasting $r_i = m_i g^{-k_i}$ by each user u_i and then computing r, in the beginning of each time period, a random value R is broadcast to the network. (This value, R, plays the role of r in the previous algorithm.) The time period is chosen such that no signatory generates more than one signature in a time period. That is, all signatures generated in time period t_j use a common parameter R_j which is broadcast by the verifier.

Signature Generation:

1. In the beginning of time period t_j, the verifier broadcasts a random value $R_j \in_R \mathbb{Z}_p$.
2. Each u_i chooses a random k_i and computes $r_i = m_i g^{-k_i} \pmod p$.
3. Each u_i generates his signature as $s_i = k_i - R'_j x_i \pmod q$, where $R'_j \equiv R_j \bmod q$.
4. u_i sends his signature (M_i, s_i, r_i) through the public channel.

Verification:

1. After receiving n signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ in time period j, the verifier
   - calculates $r_{n+1} = R_j \times (\Pi_{i=1}^{n} r_i)^{-1} \bmod p$,

- chooses a random k_{n+1} and calculates $s_{n+1} = k_{n+1} - R'_j x_{n+1} \pmod q$, where x_{n+1} is the secret key of the verifier. That is, the verifier signs a message $m_{n+1} = H(M_{n+1})$ such that $r_{n+1} = m_{n+1} g^{-k_{n+1}} \pmod p$. Note that, knowing r_{n+1} and k_{n+1}, it is easy to calculate m_{n+1}, although the verifier does not know (and does not need to know) the relevant message M_{n+1} (since the underlying hash function is one-way).

2. The verifier computes

$$m = \prod_{i=1}^{n+1} m_i \bmod p \qquad \text{and} \qquad s = \sum_{i=1}^{n+1} s_i \pmod q$$

3. The combined signature (m, s, r) is accepted if

$$m = g^s y^{R'_j} R_j \pmod p$$

Remark: The purpose of signing a dummy message by the verifier is to transform the verification of the signatures received into the general verification formula used in the proposed multisignature scheme. Note that this type of signature generation is not a security problem since the message cannot be chosen by the forgery signer. In fact, if M is chosen first, then the pair (s, r) must be calculated such that $g^s y^{r'} r$ is equal to a predetermined value. Knowing the public values of g and y and choosing one of the parameters r (or s), achieving a correct result requires solving a discrete logarithm problem for the other parameter. Considering the fact that $r' = r \bmod q$, one cannot select r' and s randomly and then solve the equation $r = H(M) \times (g^s y^{r'})^{-1}$ for calculating r.

5.4.3 Stinson Attack

After publishing [5], Stinson in [153] has shown that the Concast Scheme above is insecure. In order to illustrate the attack, let a legitimate user, u_i wish to sign a given message, M_i. To sign the message M_i, user u_i follows the algorithm given in Scheme 3 and computes the (r_i, s_i) such that:

$$H(M_i) = m_i = r_i g^{k_i} = r_i g^{(s_i + R'_j x_i)} = g^{s_i} y_i^{R'_j} r_i. \tag{5.4}$$

It is not difficult to see that anyone can generate the pair (r'_i, s'_i) that satisfies equation (5.4)

- choose a random value s'_i

- compute $r'_i = H(M_i)g^{-s'_i}y_i^{-R'_j}$
- (r'_i, s'_i) is a signature on the message M_i, since it satisfies the verification equation (5.4).

In other words, anybody can sign any message on behalf of any user.

The attack on the Concast Signature works because the calculated value r_i (by user u_i) is not determined by the parameter R'_j. In the original signature scheme, the value r, calculated in equation 5.1, is used in equation 5.2 in order to generate the signature. That is, one cannot forge a signature because it requires the discovery of a value r such that it satisfies equation 5.3, where $r' = r \pmod p$ (see [116] for more detail and formal discussion on the security of this signature scheme).

5.4.4 Secured Concast

In this section, a modification to the Concast scheme is accordingly proposed by the authors of [5] to fix this security flaw by employing the idea of *small exponent test* which is used in [15].

Small exponent test:

Let g be a generator of group G of prime order q, and $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i \in Z_q$, $y_i \in G$, and a security parameter w. The query is whether or not $y_i = g^{x_i}$ for all $i = 1, \ldots, n$.

1. Choose $w_1, \ldots, w_n \in \{0, 1\}^w$ at random.
2. Compute $x = \sum_{i=1}^n x_i w_i \pmod q$, and $y = \prod_{i=1}^n y_i^{w_i}$.
3. if $y = g^x$ then accept, else reject.

Signature Generation:

1. In the beginning of time period t_j, the verifier broadcasts a random value $R_j \in_R \mathbb{Z}_p$ and a prime security parameter $W_j = \{0, 1\}^W$.
2. Each u_i chooses a random k_i and computes $r_i = m_i g^{-k_i} \pmod p$, where $m_i = H(M_i)$ and M_i is the message in which the user u_i wishes to sign.
3. Each u_i computes $w_i = r_i \pmod{W_i}$
4. Each u_i generates his signature as $s_i = k_i - R'_j w_i x_i \pmod q$, where $R'_j \equiv R_j \bmod q$.

5. u_i sends his signature (M_i, s_i, r_i) through the public channel.

Verification:

1. After receiving n signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ in time period t_j, the verifier computes $y = \prod_{i=1}^{n} y^{w_i} i \bmod p$,

2. The verifier combines the signatures using

$$r = \prod_{i=1}^{n} r_i \pmod p, \quad m = \prod_{i=1}^{n} m_i \pmod p \qquad \text{and} \qquad s = \sum_{i=1}^{n} s_i \pmod q$$

3. The combined signature (m, s, r) is accepted if

$$m = g^s y^{R'_j} r \pmod p$$

Security Issues

Given a message M, a forger may try to find a pair (r, s) such that:

$$H(m) = g^s y^{Rr'} r,$$

where $r' = r \pmod W$ and R, W are random values chosen by the verifier (out of the signer's control). We observe the following cases:

1. if r is fixed, then computing s is equivalent to solving a discrete logarithm problem over $GF(p)$, which is believed to be a difficult problem.

2. If s is fixed, then r could be computed from the following equation:

$$r = H(M) g^{-s} y^{-Rr'} \tag{5.5}$$

Solving equation (5.5) for r, such that $r' = r \; mod \; W$, cannot be done in polynomial time. In fact, if $W = q$, then this scheme is as secure as the original scheme in [116]. The proposed scheme, however, possesses a mechanism that disrupts an attack which utilizes the choice of the small value for W. That is, the verifier chooses the time period so that solving equation (5.5) for the given R_j and W_j would not be feasible. For example, choosing a 50-bit random binary string, W, will provide reasonable security for acceptable time periods. The probability of successful heuristic attack is approximately 2^{50} and we are not aware of any other efficient way of solving equation (5.5).

Note that implementing a system which is valid for a particular time period is a well-known technique that has been applied in many other cryptographic systems. For example, proactive secret sharing [82] provides the secrecy of long-living secrets by regularly redistributing new shares of the secret (and discarding old shares) among shareholders.

3. Another possible attack could be if a forger tries to solve the equation (5.5) for both r and s simultaneously, but we are not aware of any efficient algorithm to do that.

Remark Knowing a legitimate signature of a message, it may not be possible to generate another correct signature for that message. This is not a security problem, because the message has been accepted to be signed by the legitimate signer. Moreover, knowing one or more legitimate signatures may help one to generate a sequence of signatures such that their combination can pass the screening test, but none of them (individually) can pass the verification. This is not a security problem in signature screening test (for more detail see the original paper [15]).

Performance Issues

The cost of our scheme is n applications of hash functions, $2n$ modular multiplications, and n modular additions, plus the verification of one digital signature.

The main advantage of this scheme is that there is no need for any interaction among the users. Indeed, the major shortcoming of all interactive systems is that the system must be highly synchronized. For example, in signature generation applications one cannot generate his signature before all participants have broadcast their computed value (r_i, in our protocols).

5.5 Fast Screening for a Non-RSA Signature Scheme

We first recall that screening means the task of determining whether the signer has at some point authenticated the message rather than the task of checking that the particular string is a valid signature of the message.

In [15], finding fast screening algorithms for signature schemes other than RSA has been left as an open problem. That is, instead of n signatories, a signer generates a large number of signatures and a receiver wishes to verify all these signatures (e.g., when a bank issues a large number of electronic coins and the customer wishes to verify the correctness of coins). We observe that this problem can be solved as a special case in our proposed schemes.

In ElGamal-type signatures, however, the signer must use a fresh random number for every signature; otherwise, it compromises the secrecy of the secret key of the signer. Hence, performance of the proposed schemes, which use a common random number in the generation of n different messages in a concast environment, is not acceptable in this case. In order to avoid this problem, the signer needs to follow the original signature scheme (see Section 5.3.2).

Before presenting the scheme, we establish the model of security.

5.5.1 Model of Security

A digital signature scheme $\sum$ consists of three algorithms [66]:

- GenSig, the *key generation algorithm* which, on input 1^w, where w is the security parameter, outputs a pair (x, y) of matching private and public keys;
- Sig, the *signature generation algorithm* which receives a message M and the private key x, and outputs a signature $(s, r) = \mathsf{Sig}_x(M)$;
- Ver, the *verification algorithm* which receives a candidate signature s, a message M and a public key y, and returns an answer $\mathsf{Ver}_y(M, s, r)$ as to whether or not (s, r) is a valid signature on M with respect to y. In other words, $\mathsf{V}er_{pk}(M, s, r) = 1$ would mean a valid signature to the message, whereas $\mathsf{V}er_{pk}(M, s, r) = 0$ means an invalid signature to the message.

Definition 5.1 ***A Screening test*** *means, given a sequence of instances* $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ *of* $\sum$*, then we say batch instance is correct if the test* $\mathsf{V}er_{pk}(M_i, s_i, r_i) = 1$ *for all* $i \in 1, \ldots, n$*, and incorrect if there is some* $i \in 1, \ldots, n$ *for which* $\mathsf{V}er_{pk}(M_i, s_i, r_i) = 0$.

Security Notions Attacks against a fast screening signature scheme can be classified according to the goals of the adversary. The strongest attack is called *existential forgery*, which means that the attacker can provide a single message/signature pair which passes the screening test. 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 message/signature pairs then this attack is called *known-message attack* (KMA). This list may contain messages randomly and uniformly chosen, and the attack is thus termed a *random message attack* (RMA). Finally, the message may be chosen, adaptively, by the adversary himself, and generate signatures that pass our screening test: this is a *chosen-message attack* (CMA).

5.5.2 The Screening Scheme

Sig: **(Signature Generation)**

Let x and $y = g^x$ be the secret and public keys of the signer respectively. Also, let m_i $(i = 1, \ldots, n)$ be the hash values (or any other encoding) of messages $M_1, \ldots, M_n$. In order to sign m_i, the signer performs the following steps:

1. generates a random k_i and computes $r_i = m_i g^{k_i} \pmod{p}$.
2. generates a signature on message m_i as $s_i = k_i - r'_i x \pmod{q}$, where $r'_i = r_i \bmod q$.
3. sends all signatures (M_i, s_i, r_i) to the receiver.

Ver: **(Verification)**

1. After receiving n signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$, the verifier calculates
$$r = \prod_{i=1}^{n} r_i \pmod{p}, \qquad m = \prod_{i=1}^{n} m_i \pmod{p}, \qquad \text{and} \qquad s = \sum_{i=1}^{n} s_i \pmod{q}$$
2. The verification of the combined signature (m, s, r) is the same as in the underlying signature scheme, that is, the signatures are accepted if
$$m = g^s y^{r'} r \pmod{p}$$
otherwise, they are rejected.

5.5.3 Security Analysis

The main issue in security consideration of a digital signature is to determine whether an adversary, without knowing the secret key, is able to generate a signature of a message which has never been signed by the legitimate signer but which passes the verification test; this is called *existential forgery.* This is a general question, and the answer is given in the security analysis of the underlying digital signature. (Obviously, a digital signature that allows forgery will be considered completely useless.)

In our signature screening algorithms, however, one would like to check whether it is possible to have a sequence of signatures that passes the test but contains fake signatures. We begin our security analysis of this type of attack with the following theorem.

Theorem 5.1 *Given a set, $\mathcal{S}$, consisting of n digital signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ that pass our screening test, it is impossible to find two subsets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \cap \mathcal{B} = \emptyset$, $\mathcal{S} = \mathcal{A} \cup \mathcal{B}$, and signatures in $\mathcal{A}$ (or $\mathcal{B}$) pass the test but signatures in $\mathcal{B}$ (or $\mathcal{A}$) fail the test.*

Proof. Without loss of generality, let $\mathcal{A} = (M_1, s_1, r_1), \ldots, (M_\ell, s_\ell, r_\ell)$ and $\mathcal{B} = (M_{\ell+1}, s_{\ell+1}, r_{\ell+1}), \ldots, (M_n, s_n, r_n)$, for an integer $1 \le \ell \le n$. Define

$$m_A = \prod_{i=1}^{\ell} m_i\ , \quad s_A = \sum_{i=1}^{\ell} s_i\ , \quad k_A = \sum_{i=1}^{\ell} k_i\ , \quad y_A = \prod_{i=1}^{\ell} y_i\ , \quad \text{and } r_A = \prod_{i=1}^{\ell} r_i$$

Similarly, m_B, s_B, k_B, y_B, and r_B can be defined. Note that, we have $m = m_A \times m_B$, $s = s_A + s_B$, $k = k_A + k_B$, $y = y_A \times y_B$, and $r = r_A \times r_B$. Let the sequence of signatures in the set $\mathcal{A}$ pass our screening test. The sequence of signatures in $\mathcal{A}$ forms a combined signature (m_A, s_A, r_A) such that

$$s_A = \sum_{i=1}^{\ell} s_i = k_A - r' \sum_{i=1}^{\ell} x_i \pmod q$$

and thus the verification implies that the following equation must be true

$$m_A \;=\; g^{s_A} y_A^{r'} r_A \pmod p. \tag{5.6}$$

On the other hand, the set of all signatures in the set $\mathcal{S}$ also passes the test, that is,

$$m = g^s y^{r'} r \pmod p$$

which can be written as

$$m_A \times m_B \quad = \quad g^{s_A} \times g^{s_B} \times y_A^{r'} \times y_B^{r'} \times r_A \times r_B \pmod p. \tag{5.7}$$

Now, dividing both sides of equation (5.7) by equation (5.6) gives

$$m_B = g^{s_B} y_B^{r'} r_B \pmod p$$

which indicates that the sequence of signatures in the set $\mathcal{B}$ also passes the test.

An immediate consequence of Theorem 5.1 is that:

Theorem 5.2 *If a set, $\mathcal{S} = \{(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)\}$ that passes our screening test consists of some fake signatures, then the set of all fake signatures must also pass the screening test.*

Proof. Split the sequence of signatures in $\mathcal{S}$ into two sets $\mathcal{A}$ and $\mathcal{B}$, such that $\mathcal{A}$ consists of all genuine signatures but $\mathcal{B}$ consists of all fake signatures. Using Theorem 5.1, since $\mathcal{A}$ passes the test, $\mathcal{B}$ must also pass the test.

Corollary 5.3 *Given a set $\mathcal{S}$, consisting of n digital signatures $(M_1, s_1, r_1), \ldots, (M_n, s_n, r_n)$ that passes our screening test, it is impossible that $\mathcal{S}$ contains only one fake signature. That is, either there exists no fake signature in $\mathcal{S}$ or there is more than one fake signature in $\mathcal{S}$.*

Note that, knowing a signature (M, s, r), it is easy to form a set of fake signatures that passes the screening test. For example, in order to form a set of ℓ fake signatures, one can form a set of ℓ pairs (s_i, r_i) such that $s = \sum_{i=1}^{\ell} s_i$ and $r = \prod_{i=1}^{\ell} r_i$. Clearly, this set of ℓ fake signatures $(M, s_1, r_1), \ldots, (M, s_\ell, r_\ell)$ passes our screening test. This is similar to the problem identified in [15]. We observe that it is not difficult to overcome this problem. In particular, it is easy to deal with this problem in the RSA type signatures of [15] (the RSA signature is deterministic and thus a message cannot have different signatures). That is to say, a sequence with such instances will be easily detected as faulty sequences. However, we observe another way to create a faulty sequence of signatures that passes the screening test. The method is applicable to both our schemes and the scheme introduced in [15]. Let $(M_1, \sigma_1,), \ldots, (M_n, \sigma_n)$ be a sequence of n genuine signatures. Obviously, this set passes the screening test. On the other hand, the set $(M_1, \sigma_{\pi(1)}), \ldots, (M_n, \sigma_{\pi(n)})$, where $\pi(.)$ is a random permutation over $\{1, 2, \ldots, n\}$, also passes the screening test. This means that, no matter how secure the underlying digital signatures are, it is always possible to produce a sequence of faulty signatures (in the above manner) that passes the signature screening test. However, as mentioned in [15], this is not a security problem since these attacks do not succeed without knowing the signatures of the messages, that is, the messages must be signed by legitimate signers.

Another serious threat to the scheme could be if an adversary can select messages of his own choice and then generate a (set of) signature(s) that pass(es) our screening test, which is also known as a *chosen-message attack*. The following theorem indicates the security assurance of our screening technique.

Theorem 5.4 *Let a set, $\mathcal{S}$, consist of n digital signatures (M_i, s_i, r_i), $i = 1, \ldots, n$ that passes our screening test. If the underlying digital signature is secure then $\mathcal{S}$ does not contain a message that has never been signed by a legitimate signer.*

Proof. Let $\mathcal{A} \subseteq \mathcal{S}$ and $\mathcal{A}$ consist of all messages that have never been signed by legitimate signers. Obviously, the set of all signatures in $\mathcal{A}$ passes the verification test and thus a set of unauthorized users can sign a message in a multisignature manner, which is not the case.

In multisignature schemes, if a set of unauthorized users tries to forge a signature, or when a malicious user tries to prevent the process of signature generation, the generated group signature is not genuine and it fails to pass the verification test. The following theorem presents an efficient algorithm to detect such a faulty signature (malicious user).

Theorem 5.5 *Let a set, $\mathcal{S}$, consist of n digital signatures (M_i, s_i, r_i), $i = 1, \ldots, n$ and let $\mathcal{S}$ fail to pass our screening test. There exists an $O(\log n)$ running time algorithm that detects a faulty signature (a malicious user).*

Proof. The following algorithm, which is an instance of the *binary search* algorithm, works properly, based on our results so far.

1. Split the set $\mathcal{S}$ into two subsets (with almost equal sizes) and submit one of them to the verification algorithm. If the set of signatures in this subset passes the verification test, then the other subset cannot do so (i.e. the faulty signature is in the other subset); otherwise, this set contains the faulty signature.

2. Repeat step 1 on the subset that cannot pass the verification test as long as the cardinality of the set is larger than one.

5.6 Summary

The authentication of many-to-one group communication has not been investigated in the literature. With the emergence of concast communication applications, an efficient authentication scheme and protocol is required. The essence of the problem is to have an efficient scheme that can be fast enough to verify a large number of messages instantly. Typical authentication methods such as digital signature are not suitable for the concast problem. We have proposed three different authentication schemes for this communication mode. We found that screening of batch signatures is very efficient method for such a problem. Also, we suggested a solution to the open problem of finding a fast screening signature for non-RSA digital signature schemes. A possible extension to this work, which we leave as an open problem, is to design a signcryption scheme for concast communication mode, which provides both authenticity and privacy.