A Hash-Chain Based Method for Full or Partial Authentication of Communication in a Real-Time Wireless Environment

by

Daniel Miller

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2010

Abstract

Real-time media streams are a common application on the Internet today. For many such streams, it is necessary to provide authentication, data integrity, and non-repudiation. Some applications where this type of security may be necessary include voice-over-IP (VoIP) calls, transmission of sensitive data such as medical records or personal information, or financial data that needs to be updated in real-time. It is important to be able to balance the need for security with the constraints of the environment, where data must be delivered in a limited amount of time.

This thesis examines and classifies the different types of authentication based on a number of factors, mainly the type of authentication (user or data), the way in which authentication information is transmitted (embedded or appendix), and the secrecy of the authentication information (covert or overt). This thesis then presents a specific real-time communication system, and develops a novel method of achieving data authentication for the system, based on previous work done in the area of hash-chaining authentication schemes. Theoretical and simulated results are presented, showing that the new method, the modified butterfly scheme, outperforms the original method, the butterfly scheme, using the same amount of overhead.

Acknowledgements

I would like to thank my supervisors, Dr. Gord Agnew and Dr. Andrew Morton, as well as my thesis committee members, Dr. Ian Munro and Dr. Bill Bishop, for their help in preparing this thesis. I would also like to thank Ali Ahmadzadeh for his help on the modified butterfly-graph based hash-chaining method, and for writing the Matlab simulation code.

Dedication

To my fiancée, Chiara

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real-time media streams are a common application on the Internet today. For many such streams, it is necessary to provide authentication, data integrity, and non-repudiation. Some applications where this type of security may be necessary include voice-over-IP (VoIP) calls, transmission of sensitive data such as medical records or personal information, or financial data that needs to be updated in real-time. It is important to be able to balance the need for security with the constraints of the environment, where data must be delivered in a limited amount of time.

This thesis examines and classifies the different types of authentication, starting with the fundamental division between user authentication and data authentication. This thesis then presents a specific real-time communication system, and looks at the best way to perform data authentication for this particular system. A modification of an existing data authentication technique (signatures with hash-chaining) is presented, and the modified technique is compared to the original.

The remainder of this thesis is organized as follows. Chapter 2 presents some background on authentication methods, and classifies them based on various aspects. Chapter 3 examines some specific authentication systems that fall into different categories in the classification. Chapter 4 presents a real-time communication system, examines some techniques for providing data authentication, and chooses a method to give the best tradeoff between security and the requirements of the real-time environment. Chapter 5 provides conclusions and areas for future work.

Chapter 2

Classification of Authentication Systems

Broadly, authentication can be divided into two main functions. The first is user authentication - that is, ensuring that a user actually possesses the identity that he or she claims to possess. The second is data authentication - preventing the unauthorized addition, deletion, or modification of a message.

2.1 Data Authentication

A key concept in data authentication is the idea of a digital signature. That is, a sender can provide some information (a signature) along with a message that allows the recipient to verify that it has not been modified in transit. Menezes *et al* [1] divide digital signature schemes into two categories: signature with appendix and signature with message recovery. Appendix means that the signature and original message are sent separately, and the original message is required as an input to the verification function, as in Figure 2.1.

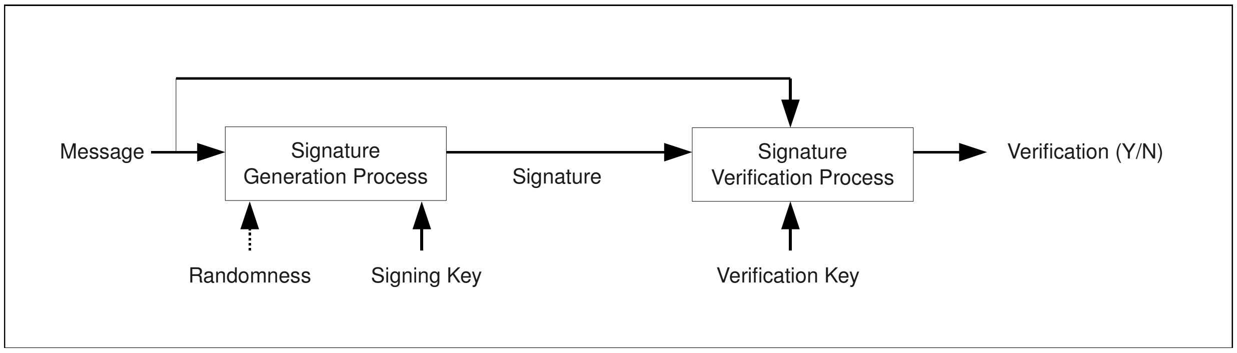


Figure 2.1: General model of an appendix signature scheme.

Message recovery means that the message can be obtained from the signature itself, so it is not required to have the original message in order to verify the signature. This classification of signatures with message recovery can be expanded to a class that, in this thesis, will be called "embedded" data authentication. This refers to any technique where the message and signature are sent as an inseparable unit, as in Figure 2.2; that is, there is no need to send the plaintext message separately from the signature component. This covers both signature schemes with message recovery, as well as signatures that are embedded in the message data in some way, as in watermarking systems.

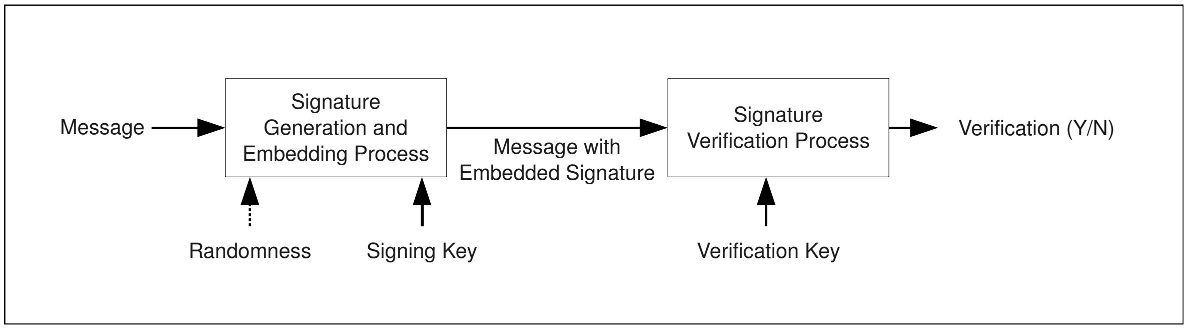

Message
Signature Generation and Embedding Process
Message with Embedded Signature
Signature Verification Process
Verification (Y/N)
Randomness
Signing Key
Verification Key

Figure 2.2: General model of an embedded signature scheme.

Figure 2.3 shows a categorization of the different mechanisms for data authentication. Categories are shown in white, while specific examples are shown in grey.

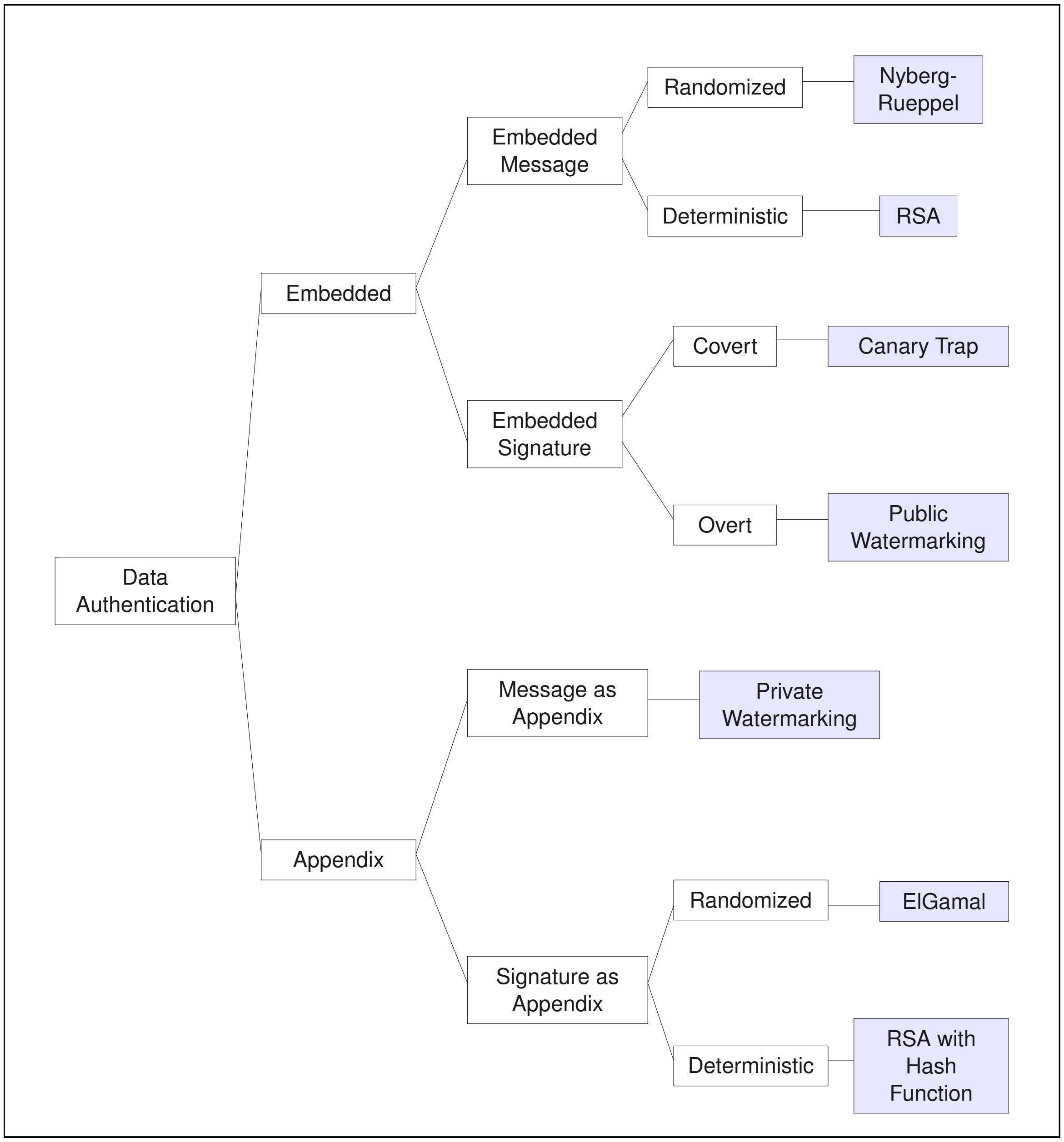


Figure 2.3: Classification of data authentication schemes.

2.1.1 Embedded Data Authentication

Embedded authentication schemes can be divided based on what information is being embedded. Some techniques, like steganography or digital watermarking, embed some authentication data in a message. In this thesis, this type of system will be referred to as embedded signature. Other techniques, such as RSA signatures, embed a message within a signature. These types of systems will be referred to as embedded message.

2.1.1.1 Embedded Message

In an embedded message scheme (also known as digital signature with message recovery), the sender computes a signature of the message and sends it to the receiver. This is generally done with a public key encryption system like RSA, which will be discussed later. The receiver is then able to both verify the identity of the sender and extract the message from the signature.

Digital signature schemes with message recovery can be either randomized or deterministic. In a deterministic scheme, a given message and key will always produce the same signature. In a randomized scheme, a given message and key can produce multiple signatures, depending on a redundancy function that is applied to the message before it is signed.

In practice, signature schemes with message recovery are rarely used, for two reasons. First, the signature size is directly correlated to the message size. This means that the size of the message that can be signed is limited, because the signature of a large message is often expensive to compute. Second, it is easy to mount what is known as an "existential forgery" attack, by sending a random string as the signature. Of course, the attacker has no control over what the corresponding message will be, but the receiver will still treat it as a valid message. It is desirable to prevent this type of attack, even though it is one of the weakest forms of attack on a signature scheme.

2.1.1.2 Embedded Signature

In an embedded signature scheme, some information (a signature) is embedded in the message to identify the sender and content. Usually this is done in such a way as to be imperceptible to someone consuming the message (especially for messages containing audio or video data), but this is not always the case. The embedding of the signature can be done either covertly or overtly.

The aim in a covert embedded signature scheme is to prevent the attacker from even knowing that the authentication data exists. This type of system falls under the domain of steganography. Steganography, from the Greek for "covered writing" [2], is the technique of hiding not only a message, but the fact that the message exists at all. Generally, in such a covert scheme, the focus is on concealing the signature's existence, rather than making it robust to attacks or attempts to remove it.

In an overt embedded signature scheme, the attacker (and everyone else) is aware that the authentication data is embedded in the message. Therefore, to ensure data integrity, the goal is to make it impossible to remove or alter the authentication data without destroying the message. This type of system is a form of digital watermarking. If the original, unmarked message is not required as an input to the verification function (as in the case of an embedded signature), then the scheme is referred to as a public watermarking system.

2.1.2 Data Authentication with Appendix

Data authentication with appendix is differentiated from embedded data authentication by the fact that it requires the original message in order to verify the authentication information. It can be further divided based on what data is sent as the appendix: the signature, or the message itself.

2.1.2.1 Signature as Appendix

This is the most commonly-used type of data authentication scheme. A signature of the message is computed, and is then sent along with the message itself. Using the message and the signature, the receiver can verify the data. This type of scheme typically relies on a hash function, which means it can be applied to messages of any length. Like signatures with message recovery, signatures sent as appendices can be either randomized or deterministic. It is possible to construct an appendix scheme from any message recovery scheme simply by hashing the message before signing it [1].

In a deterministic digital signature scheme with appendix, a hash of the data is computed, using some well-known hashing algorithm like SHA-1 [3]. In this way, the signature will be a fixed length, rather than varying with the size of the message. Then, the hash will be signed, that is, encrypted with the sender's private key. The signed hash is sent to the receiver along with the original message.

The receiver can compute the hash of the message, then decrypt the signed hash with the sender's public key, and verify that they match. This ensures data integrity: if the message or the signed hash are modified in any way, they will not match. If a secure hashing algorithm is chosen, it should be computationally infeasible to modify the message and signature in such a way that they will still match even after the modifications.

Randomized appendix signature algorithms, like their embedded counterparts, can produce different signatures for the same message-key pair. Some examples include ElGamal signatures [4] and related schemes like the Digital Signature Algorithm (DSA) [5].

2.1.2.2 Message as Appendix

It would seem counter-intuitive to send a message as an appendix to the message itself. However, this category encompasses mechanisms like private watermarking, where the sender sends a watermarked version of the message, but the receiver must have a copy of the original unmarked message in order to extract the watermark. The unmarked message can be exchanged through a secure channel.

2.2 User Authentication

User authentication can be divided into real-time authentication and transfer-of-trust. The goal in real-time authentication is to confirm that a user actually has the identity that he or she claims. It typically involves a challenge and response (such as prompting a user for their password). Transfer-of-trust means that a trusted third party vouches for the user. This is often expressed by having the trusted third party issue a certificate to the user, confirming his or her identity. The trust in the third party implicitly guarantees the authenticity of the user. These two methods can be complementary to one another. For example, a system could use real-time authentication to identify a user at the beginning of a session, and then issue him or her a certificate to be used for the duration of the session.

Figure 2.4 shows a categorization of the different mechanisms for user authentication. Categories are shown in white, while specific examples are shown in grey.

2.2.1 Real-Time User Authentication

Real-time user authentication can be broken down into three different factors, informally known as “something you have,” “something you know,” and “something you are” [6]. Token-based authentication (something you have) is normally a physical device such as a USB dongle or a security card. Knowledge-based authentication (something you know) could be a password, or it could involve questions about the user’s life. Anyone who has used online banking has likely been asked to set up several security questions such

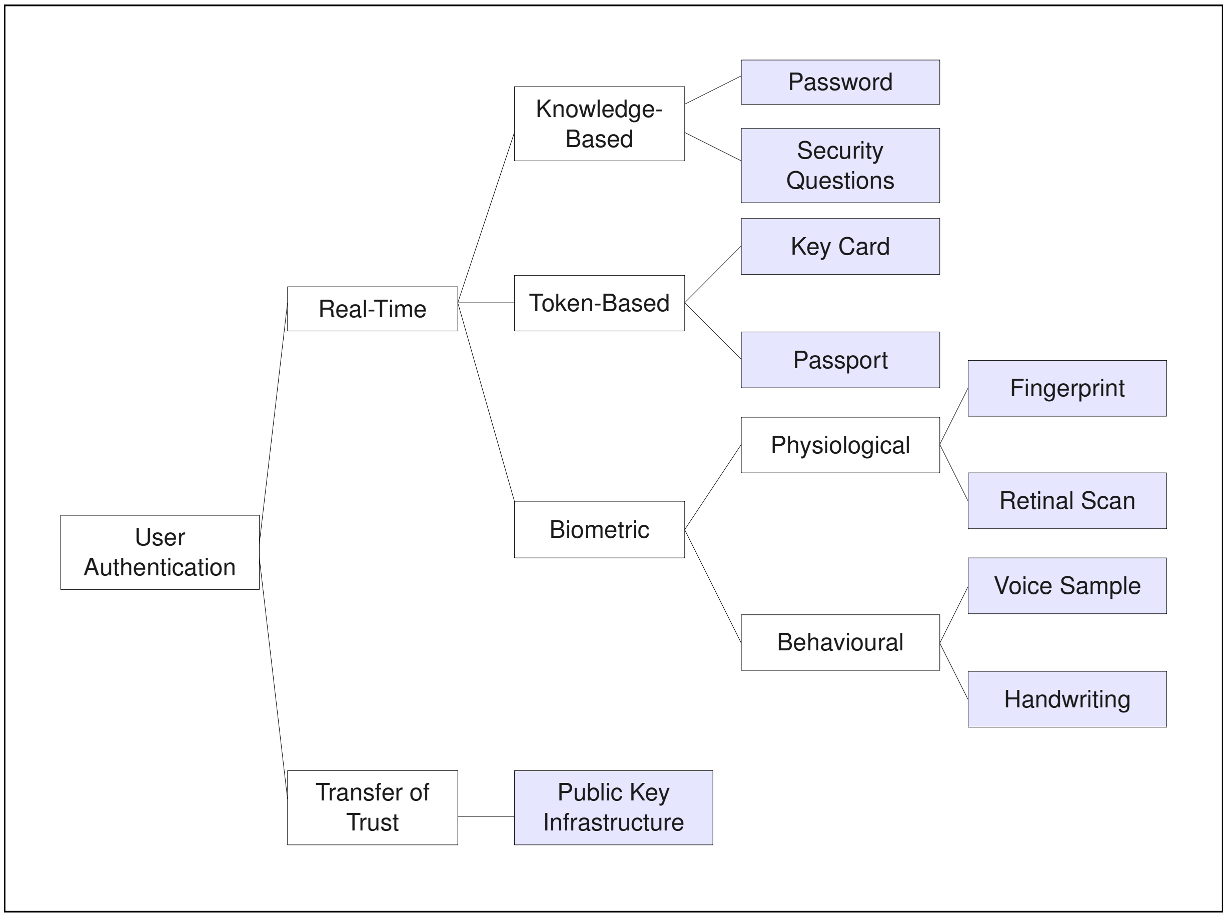


Figure 2.4: Classification of user authentication schemes.

as "Mother's maiden name" or "Father's middle name." Biometrics (something you are) can be further divided into physiological traits, like a fingerprint or retinal scan, and behavioural traits, such as a handwritten signature or a voice sample [7]. Behavioural traits may vary from sample to sample, whereas physiological traits are more or less constant.

2.2.2 Transfer-of-Trust

A transfer-of-trust system requires a third party that is trusted by the entity attempting to authenticate the user. One basic example is a public key directory. When an entity receives a copy of a user's public key, it must have a way to ensure that that key actually belongs to the user in question, and not to an attacker. To accomplish this, all users can register their public keys with a central directory, where they are signed with the directory's public key. The directory's public key is known to everyone. An entity can then obtain a user's public key from the directory, and confirm its authenticity by decrypting it with the directory's public key. In this case, the directory is the trusted third party. More complex hierarchies can be developed based on this simple approach, in order to improve performance.

Chapter 3

Examples of Authentication Systems

3.1 RSA Signatures

Rivest, Shamir, and Adleman [8] proposed the first digital signature scheme, RSA, in the same paper where they proposed the first public-key encryption system. RSA is a deterministic embedded message scheme. Like RSA encryption, RSA signatures are based on the integer factorization problem. Each person who wants to create a signature performs the following procedure. First, he chooses two large primes, p and q. He computes $n = pq$. The product n is public knowledge, although p and q are not. He then chooses his private key, d, and public key, e, such that:

$$ed \equiv 1 \bmod (p-1)(q-1) \tag{3.1}$$

The private key is kept secret, and the public key is made available to all users of the system. To sign a message m, the sender computes:

$$s \equiv m^d \bmod n \tag{3.2}$$

The recipient must calculate:

$$m \equiv s^e \bmod n \tag{3.3}$$

thereby recovering the message and verifying both its source and its integrity. Because the message need not be sent separately, this system provides message recovery.

In practice, the public key e is generally chosen to speed up the verification calculation. Common values are $e = 3$ and $e = 2^{16} + 1$ [1]. The private key d is then calculated using Equation 3.1. If d is chosen instead, and it is selected to be too small, then it is possible to use an efficient algorithm to determine d from n and e, as shown by Wiener [9]. "Too small" in the context of Wiener's attack means, approximately, $d \leq \frac{n}{4}$. To avoid this attack, it is generally recommended that d be approximately the same size as n [1].

If n can be easily factored, then an attacker can compute d with the public information n and e. It is therefore important to choose the primes p and q in such a way that it is computationally infeasible to factor n. The difference $p - q$ should not be too small; otherwise, $p \approx q \approx \sqrt{n}$, which makes it easy to factor n [1]. In addition, $p - 1$ and $q - 1$ should each have a large prime factor. If they do not, then n can be factored using Pollard's $p - 1$ factoring algorithm, described in [10].

The RSA signature system is also vulnerable to an existential forgery attack - that is, an attacker can easily create a valid message and signature pair, although not necessarily of his choosing. To do so, he simply chooses a signature at random and decrypts it with the sender's public key to get the corresponding message. To thwart this type of attack, the message can be hashed before signing it. However, this means that the system no longer provides message recovery, making it a deterministic signature as appendix scheme.

3.2 ElGamal Signatures

The ElGamal signature scheme [4] is a randomized signature as appendix scheme. All users in the system must share p, a large prime, and α, a primitive element mod p. Additionally, each user must have a private key, x, and a public key, $y \equiv \alpha^x \bmod p$.

To sign a message m, a user chooses a random number k between 0 and $p-1$, such that $\gcd(k, p-1) = 1$. He then computes:

$$r \equiv \alpha^k \bmod p \tag{3.4}$$

and calculates s such that:

$$m \equiv xr + ks \bmod p \tag{3.5}$$

He can then send the message m along with the signature (r, s).

To verify the signature, the recipient checks that:

$$\alpha^m \equiv y^r r^s \bmod p \tag{3.6}$$

3.3 Nyberg-Rueppel Signatures

Nyberg and Rueppel [11] proposed a method of using the National Institute of Standards and Technology's Digital Signature Algorithm (DSA) [5] to create a signature with message recovery. In the terminology of this thesis, the Nyberg-Rueppel signature is a randomized embedded message scheme. The DSA is itself based on the ElGamal signature scheme. Nyberg and Rueppel further extended their method to add message recovery to any signature scheme based on ElGamal encryption [12].

In Nyberg and Rueppel's original algorithm, all users must share a large prime p, and an element α of order $p-1$. In addition, each user must have a private key, x, and a public key, $y \equiv \alpha^{-x} \bmod p$. To sign a message m, a user randomly generates a number k. He then computes:

$$e \equiv m\alpha^{-k} \bmod p \tag{3.7}$$

and:

$$w \equiv k + xe \bmod p \tag{3.8}$$

The signature is the pair (e, w).

To verify the signature and recover the message, the recipient computes:

$$m \equiv \alpha^{w} y^{e} e \bmod p \tag{3.9}$$

3.4 Canary Trap

One example of a covert embedded signature scheme is known as a "canary trap." It is useful for detecting the source of an information leak. In this scheme, several similar copies of a document are prepared. They all differ slightly in their wording (using several pairs of synonyms, for example). If a leaked version of the document appears, it can be traced back to the source by the particular wording that was used. The name "canary trap" comes from Tom Clancy's novel *Patriot Games*. The following is the section of that book that describes the canary trap in detail [13]:

> *"Each summary paragraph has six different versions, and the mixture of those paragraphs is unique to each numbered copy of the paper. There are over a thousand possible permutations, but only ninety-six numbered copies of the actual document. The reason the summary paragraphs are so – well, lurid, I guess – is to entice a reporter to quote them verbatim in the public media. If he quotes*

something from two or three of those paragraphs, we know which copy he saw and, therefore, who leaked it. They've got an even more refined version of the trap working now. You can do it by computer. You use a thesaurus program to shuffle through synonyms, and you can make every copy of the document totally unique."

3.5 Digital Watermarking

Digital watermarking refers to "the imperceptible, robust, secure communication of information by embedding it in and retrieving it from other digital data" [14]. Rather than a specific technique, digital watermarking encompasses a set of tools that are useful for particular applications. An in-depth discussion of the entire field of digital watermarking is outside the scope of this thesis; however, an overview of the relevant applications is presented. Authentication is a common application; that is, confirming the source of a piece of data, or ensuring that it has not been modified. A related application is copyright protection, which includes identifying and proving the owner of the data.

An overview of the watermark embedding process is shown in Figure 3.1. The corresponding watermark detection process is shown in Figure 3.2.

Digital watermarking systems are classified in a number of ways. One of the most common is whether the original data is required in order to detect or extract the watermark. If the original data is necessary, the system is classified as private watermarking. This is considered a message as appendix scheme, since the original message must be sent along with the "signature," the watermarked message. If the original data is not required, it is public watermarking. This falls under the category of overt embedded signature schemes.

Watermarks can also be classified based on robustness. Some watermarks, known as fragile watermarks, are sensitive to any changes in the original data. Other watermarks are

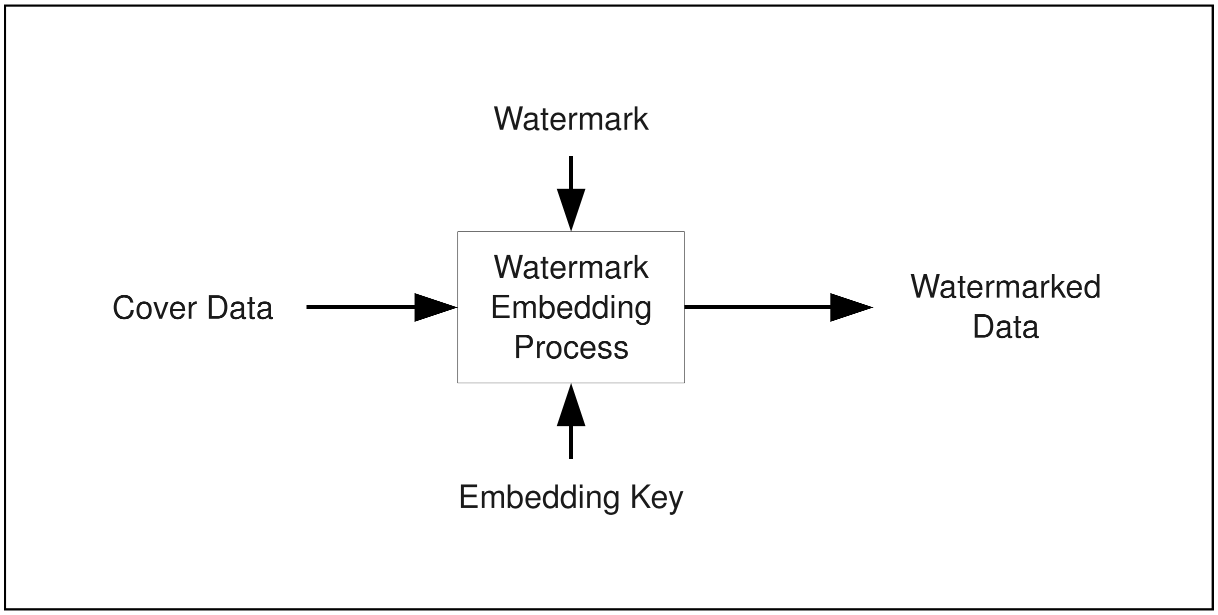


Figure 3.1: General watermark embedding process.

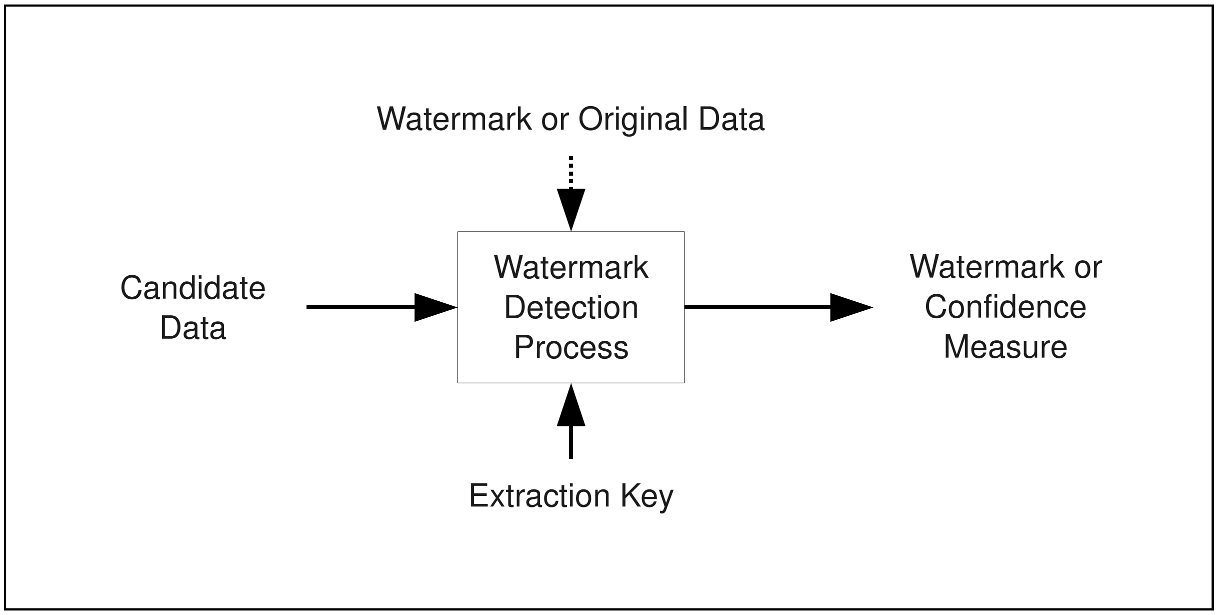

Watermark or Original Data
Candidate Data
Watermark Detection Process
Watermark or Confidence Measure
Extraction Key

Figure 3.2: General watermark detection process.

designed to survive certain transformations of the data, such as compression, filtering, or format changes. Fragile watermarks are useful for detecting any changes in the data. Nonfragile watermarks are useful in applications like copyright protection, where it is essential that the watermark cannot be easily separated or removed from the cover data.

In contrast with steganography, in a digital watermarking system, it is generally known that a watermark is embedded in the cover data. Therefore, the main attacks against watermarks are to attempt to remove them or to corrupt them in such a way that they are no longer recognizable. For this reason, a secure watermarking system should be robust against these types of attacks. All practical watermarking systems use a key, or set of keys, to ensure that only a legitimate user may embed or remove a watermark [2].

Chapter 4

An Implementation of a Data Authentication System

In this chapter, a specific implementation of a data authentication system is presented. The system is a secure, authenticated, peer-to-peer voice-over-IP (VoIP) transceiver.

4.1 Overview of the System

The secure authenticated transceiver (SAT) consists of two prototype devices that communicate with each other over the Internet, sending compressed audio data back and forth. The system also encompasses several servers to manage public key certificates, and a directory server to store the IP address of each device. The devices each have a hardware certificate, and users are bound to a device for a certain length of time with a binding certificate. Each certificate server also has its own certificate, used to identify itself to other servers. Binding certificates are used to authenticate users and devices, and to exchange a session key that is used to encrypt each conversation. All public key certificates use elliptic curve cryptography (ECC). Data is encrypted using the Advanced Encryption Standard

(AES). The goal is to add a mechanism for non-repudiation, so that a conversation may be stored (with signature) and authenticated after the fact.

One major concern with this system is that it handles real-time traffic, so the encryption and signing process must be fast enough to avoid a noticeable end-to-end delay. It is also desirable to minimize the power consumption of the prototype devices. Finally, there is the issue of packets being lost or corrupted, which could affect the ability to verify signatures at the receiving end. This is especially a concern in wireless environments. Note that if a packet is lost, there is no attempt to retransmit it - it is simply dropped. Therefore, it is desirable to sign the data in such a way that it can still be verified even if some percentage of the packets are missing - in other words, a measure of partial authentication is necessary.

4.2 Proposed Solutions

A first attempt at a solution would be to sign a hash of every data packet, and send this signature along with the data (a signature as appendix scheme). If a packet or its signature is lost in transit, it is simply ignored. As long as only a small percentage of the signatures are lost in this way, it should be possible to reconstruct the authenticated conversation with reasonable fidelity - there will be a small amount of noise where packets are missing or unverified. However, there is a problem with this approach. If a data packet or its signature is received with only a single bit in error, the signature will not match, and the whole packet will have to be thrown out. If the bit error rate is high (as in a wireless channel), there could be very few authenticated packets. It may also not be computationally feasible to sign every packet, as this would increase the time and power consumption of the application.

Another possible solution is a content-based signature, as described by Schneider and Chang [15]. Such a signature would allow small changes in the data, assuming that the changes do not affect the overall content of the voice data. This could potentially allow the system to handle small amounts of noise in the transmission channel, while still protecting against malicious attacks. A disadvantage of this approach is that it is very data-dependent – that is, different content-extraction functions would be needed for audio, video, images, and other types of data.

A third approach is a hash-chaining mechanism, which allows one signature to authenticate multiple packets. In the most basic such scheme, the hash of packet P_i is computed and appended to packet P_{i+1}. Then, if P_{i+1} is authenticated, so is P_i. This is done for every packet, so that the packets form a chain, and the final packet in the chain is hashed and signed. A simple hash-chain is shown in Figure 4.1. In this way, a single signature can be used to verify the entire chain. Variations on the basic hash-chaining method can provide better performance. This method is content-independent, and has a much lower cost compared to signing each packet individually. In addition, it is possible to achieve some measure of partial authentication, even if some packets are not able to be verified. Therefore, the hash-chaining approach was chosen for the SAT project.

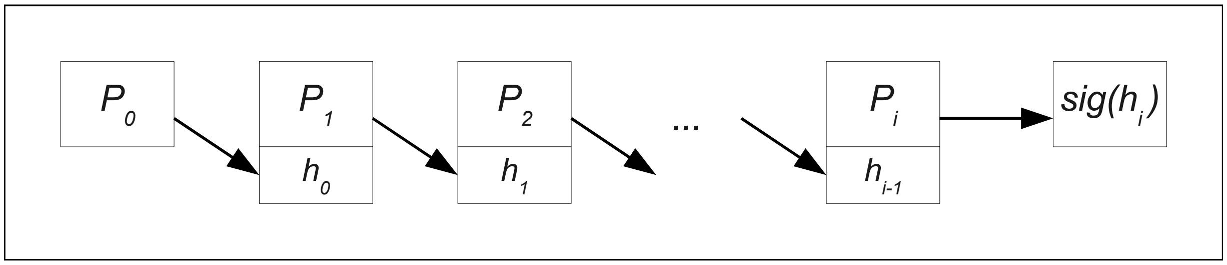


Figure 4.1: A simple hash-chaining scheme.

4.3 Existing Hash-Chaining Schemes

In this section, some existing hash-chaining schemes are explored in more detail. A method called butterfly hash-chaining is also described, which will be the foundation for the development of a new hash-chaining system to be used in the SAT project.

4.3.1 Simple Hash-Chaining

Gennaro and Rohatgi [16] first proposed a simple hash-chain, as described above, where each packet's hash is appended to the next, and the final packet is hashed and signed. This reduces the number of signatures needed to authenticate a data stream, but does not address the issue of packets that are lost in transit. So, if one packet in the chain is lost or corrupted, then no packets after that can be authenticated.

A hash-chaining system can be represented as a directed graph, where each node represents a packet, and each edge represents a hash being appended to a packet. This is shown in Figure 4.2. If a packet is lost, it and all its edges are removed from the graph. To authenticate a packet, there must be a path from that packet to the signature node. It is possible to deal with packet loss by appending each hash to more than one packet. This means that there will be multiple paths from a given packet to the signature, so that if a node on one path is lost, the packet may be verified through a different path.

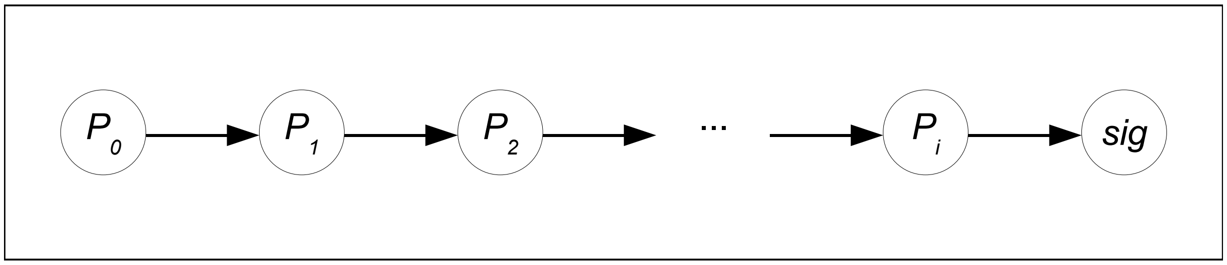


Figure 4.2: A simple hash-chain as a directed graph.

4.3.2 Efficient Multi-Chained Stream Signature

Perrig *et al* [17] proposed a method called Efficient Multi-chained Stream Signature (EMSS). In EMSS, the authors experiment with appending anywhere from two to six hashes to each packet. The hashes are appended based on a vector. For example, [2, 3, 5] means that the hash of packet P_i is appended to packets P_{i+2}, P_{i+3}, and P_{i+5}. All vectors up to length 6 were simulated to see which would give the best performance (the highest probability of verifying a packet). It was determined that [5, 11, 17, 24, 36, 39] was one of the strongest vectors. Signatures are sent at regular intervals, with up to 1000 packets sent between signatures. This scheme can tolerate a high degree of packet loss, but requires significant overhead - an extra six hashes per packet.

4.3.3 Augmented Hash-Chaining

Golle and Modadugu [18] presented what they call an augmented hash-chain. This scheme allows some buffering of packets at the sender, so that it is possible to append the hash of a packet P_i to a packet P_{i-a} preceding it. It is designed to resist bursty packet loss, which is a common model of packet loss over the Internet. Golle and Modadugu's scheme adds a maximum of five hashes to any given packet, averaging two hashes per packet.

4.3.4 Star Chaining and Tree Chaining

Wong and Lam [19] proposed two techniques, tree chaining and star chaining. In their paper, they presented star chaining first, and then generalized it to tree chaining.

In star chaining, packets are grouped into blocks. Within a block, each packet is hashed to produce a digest. The packet digests are then assembled and hashed to produce a block digest, as in Figure 4.3. In this figure, D_1 through D_8 are the packet digests, and D_{1-8} is the block digest. The block digest is signed and transmitted along with the packets themselves.

To verify a packet at the receiver, some additional information, called the packet signature, is required. In this case, the packet signature consists of the digests of all other packets in the block. Using the example of Figure 4.3, when the receiver gets packet 3, he can compute its digest, then combine it with the other packet digests to compute the block digest, and thereby verify the packet. Digests can be cached, so that when the next packet is received, the receiver only needs to verify its digest against the cached digest for that packet.

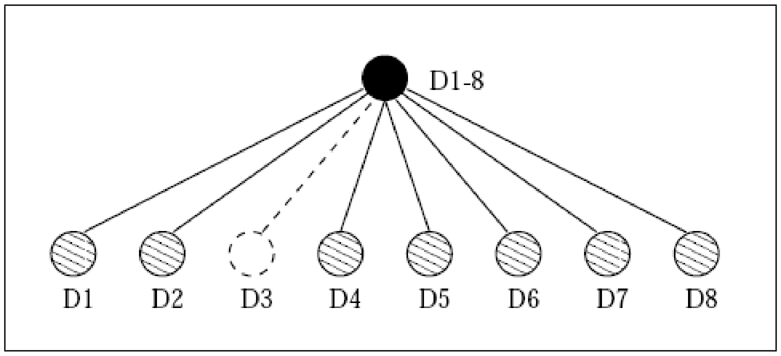


Figure 4.3: Example of star chaining [19].

Obviously, star chaining requires a lot of overhead, since each packet must carry the digests for all other packets. Tree chaining attempts to improve upon this by reducing the size of each packet signature. In tree chaining, the packet digests are grouped into a tree, as in Figure 4.4. The leaf nodes of the tree are the packet digests. All non-leaf nodes are computed by hashing the node's children together. The root node is the block digest, which is signed and sent along with the packets. A packet signature in tree chaining consists of the siblings of each node in the packet's path to the root. For example, in Figure 4.4, the signature for packet 3 would consist of D_4, D_{1-2}, and D_{5-8}.

4.3.5 Butterfly Hash-Chaining

Zhang *et al* [20] proposed a butterfly-graph based hash-chaining scheme. In the butterfly scheme, packets are grouped into blocks, and one signature is generated per block. Each block is also divided into "stages" containing a certain number of packets. The hashes of

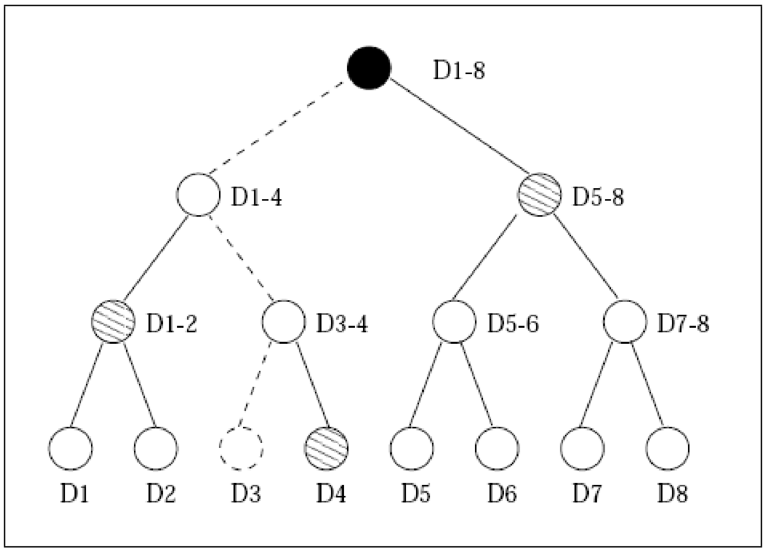


Figure 4.4: Example of tree chaining [19].

the packets in each stage are appended to the packets in the following stage. If there are N packets in a stage, then the number of packets in a block is $T = N((\log_2 N) + 1)$, and the number of stages is $M = (\log_2 N) + 1$. This makes the scheme somewhat limited, in that it does not allow blocks or stages of arbitrary size. One signature is generated on the hashes of all the packets in the final stage of each block. It is assumed that the signature packet is received correctly. This can be guaranteed by, for example, retransmitting the signature packet if it is lost.

The name of the scheme comes from the so-called "butterfly" pattern used to decide which hashes are appended to which packets. It works as follows. The function $\gamma^y(x)$ is defined; its value is obtained by flipping the yth bit of x. Stages are numbered from $M-1$ to 0, and packets within a stage are numbered from 0 to $N-1$. If $P_{m,n}$ is packet number n in stage m, then the hash of $P_{m,n}$ is appended to both $P_{m-1,n}$ and $P_{m-1,\gamma^{M-1-m}(n)}$. Figure 4.5 shows the directed graph corresponding to the butterfly authentication method, with 4 stages and 8 packets per stage, for a total of 32 packets per block. S is the signature of the hashes in the final stage.

As shown in Zhang *et al*'s paper, the authentication probability for the packets in the first stage (the one farthest from the signature packet) is:

$$\Phi_{min} = (1 - p^2)^{\log_2 N} \tag{4.1}$$

where p is the probability of losing a packet, assumed to be equal and independent for all packets.

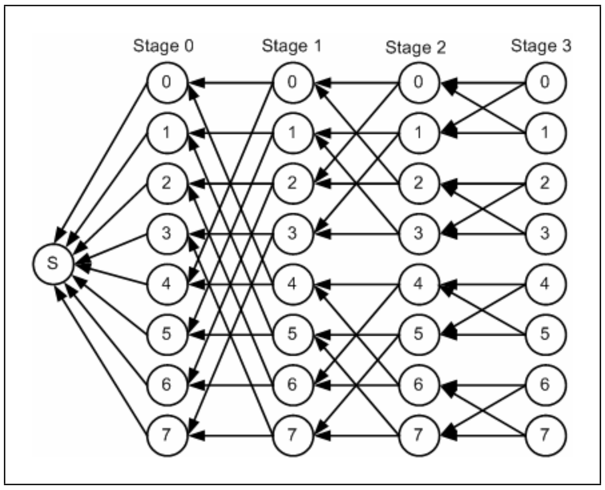


Figure 4.5: Example structure of a butterfly graph [20].

This scheme can tolerate packet loss, and performs better than other hash-chaining schemes with the same amount of overhead.

4.4 Modified Butterfly Scheme

The modified scheme that was developed for the SAT project is based on the butterfly method. In the modified butterfly scheme, each hash is split into two halves. Some of these half-hashes are XORed with others. The resulting hash segments are then rearranged and appended to the packets of the next stage according to a certain algorithm. In this way, a packet can be partially authenticated even if part of its hash is lost or received incorrectly. The modified method outperforms the original butterfly method using the same amount of overhead.

4.4.1 Basic Operation

As in Zhang *et al*'s scheme, the stream is divided into blocks of T packets. Each block is divided into M stages of N packets each, where $N = 2^{2(M-1)}$, and $T = M \times 2^{2(M-1)}$. The stages are numbered from 0 to $M - 1$, and the packets in each stage are numbered from 0 to $N - 1$. Note that the stages are numbered in reverse order as compared to the original butterfly scheme (the final stage is $M - 1$ instead of 0).

In stage m, for $0 \leq m < M - 1$, the hashes of all packets in the stage are computed and each hash is divided into two halves. The half-hashes will be combined in a particular way to form what are called "hash segments." Each segment will be the same size as one half-hash. Four such segments will be appended to each packet in the next stage, $m + 1$, as shown in Figure 4.6. That way, the overhead of the modified scheme is the same as the overhead of the original butterfly scheme. The $s_{m+1,k}$ in Figure 4.6 are the hash segments appended to the packets in stage $m+1$. There are $4N$ such segments in every stage (except stage 0), numbered from 0 to $4N - 1$. The function C in Figure 4.6 is a permutation and combination of the half-hashes, which will be described later in more detail.

Once the half-hashes are computed for stage m, they are assembled into a vector H_m of length $2N$.

$$H_m = \begin{bmatrix} h^1_{m,0} & h^2_{m,0} & h^1_{m,1} & h^2_{m,1} & \dots & h^1_{m,N-1} & h^2_{m,N-1} \end{bmatrix} \tag{4.2}$$

$h^1_{m,n}$ (or $h^2_{m,n}$) refers to the first (or second) half of the hash of packet $P_{m,n}$. Below is an

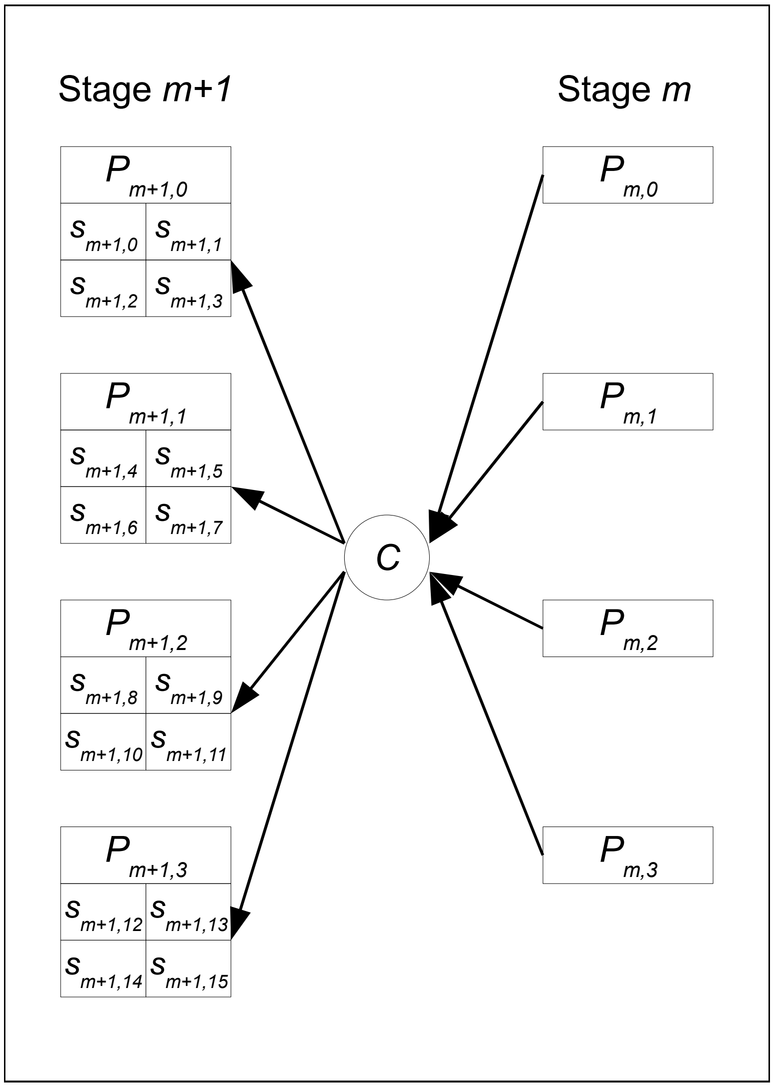


Figure 4.6: Hash segments appended to each packet, for N=4.

example for $N = 4$.

$$H_0 = \begin{bmatrix} h^1_{0,0} & h^2_{0,0} & h^1_{0,1} & h^2_{0,1} & h^1_{0,2} & h^2_{0,2} & h^1_{0,3} & h^2_{0,3} \end{bmatrix}$$

The goal is to generate a matrix K_{m+1} to indicate which half-hashes from stage m will be combined to form the hash segments for stage $m+1$. It is then possible to obtain S_{m+1}, a vector of all the segments for stage $m+1$, as follows:

$$S_{m+1} = H_m \times K_{m+1} = \begin{bmatrix} s_{m+1,0} & s_{m+1,1} & \dots & s_{m+1,4N-2} & s_{m+1,4N-1} \end{bmatrix} \tag{4.3}$$

However, having one large matrix for the entire stage is rather unwieldy, and does not scale well as N becomes large. Therefore, it was decided to break up the matrix K_{m+1} into several smaller matrices.

For each packet in stage $m+1$, a "local kernel matrix" $K_{m+1,n}$, of dimension $2N \times 4$, is generated. The local kernel matrix will select the half-hashes to be appended to that packet, and can be generated on the fly using a set of simple equations. Each row of $K_{m+1,n}$ corresponds to one of the half-hashes in the previous stage, m. A one in a particular row selects that half-hash. Each column of $K_{m+1,n}$ produces one of the hash segments $s_{m+1,k}$ to be appended to the packet $P_{m+1,n}$, as in Figure 4.6. Thus, there are four columns, since four segments are appended to each packet. The combining operation is an XOR, rather than simple addition. An example matrix is shown below, for $N = 4$.

$$K_{1,0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.4}$$

Multiplying the vector H_m by a packet's local kernel matrix yields another vector $S_{m+1,n}$

of length 4, containing the four segments that are to be appended to packet $P_{m+1,n}$.

$$S_{m+1,n} = H_m \times K_{m+1,n} = \begin{bmatrix} s_{m+1,n} & s_{m+1,n+1} & s_{m+1,n+2} & s_{m+1,n+3} \end{bmatrix} \tag{4.5}$$

For example, using the local kernel matrix above yields the following vector (the $\oplus$ operator represents XOR):

$$\begin{aligned} S_{1,0} &= H_0 \times K_{1,0} \\ &= \begin{bmatrix} h^1_{0,0} & h^2_{0,0} \oplus h^2_{0,1} & h^1_{0,2} & h^2_{0,0} \oplus h^2_{0,2} \end{bmatrix} \\ &= \begin{bmatrix} s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \end{bmatrix} \end{aligned}$$

In the final stage, $M - 1$, all the packets are hashed, and the hashes are concatenated. The concatenation of the hashes is signed and sent to the receiver. In this way, only one signature needs to be sent for each block of T packets.

A packet is verifiable if there is a path from that packet to the signature. In the modified butterfly scheme, a packet is only removed from the graph if neither one of its half-hashes can be verified. So, as long as at least one half of a packet's hash can be verified, that packet may be used to verify other packets. If a higher degree of confidence is desired, a packet can be removed from the graph if either of its half-hashes cannot be verified.

4.4.2 Generation of Local Kernel Matrices

The local kernel matrices are generated as follows. $K^{i,j}_{m,n}$ is defined to be the entry at row i and column j of the local kernel matrix of packet n in stage m. Note that i and j are indexed starting from 0. The equations in Table 4.1 are used to determine which of the $K^{i,j}_{m,n}$ are equal to one. All other entries in the matrix are equal to zero. No local kernel matrices are required for stage 0, because no hash segments are appended to the packets in stage 0. The function $\gamma^y(x)$ is the bit-flipping function described in Section 4.3.5.

Table 4.1: Equations for calculating local kernel matrices.

i	**j**
$2n$	0
$2n+1$ $2\gamma^{2m-2}(n)+1$	1
$2\gamma^{2m-1}(n)$	2
$2n+1$ $2\gamma^{2m-1}(n)+1$	3

For example, to generate the matrix $K_{1,0}$ in Equation 4.4, simply substitute $m = 1$ and $n = 0$ in the equations in Table 4.1. This yields the coordinates shown in Table 4.2.

Table 4.2: Example matrix calculation for m=1, n=0.

i	**j**
0	0
1	1
3	1
4	2
1	3
5	3

Thus:

$$K_{1,0}^{0,0} = K_{1,0}^{1,1} = K_{1,0}^{3,1} = K_{1,0}^{4,2} = K_{1,0}^{1,3} = K_{1,0}^{5,3} = 1$$

All other $K_{1,0}^{i,j}$ are equal to zero.

4.5 Analysis and Results

In this section, the modified butterfly scheme is examined in more detail. A number of factors are discussed, including the verification probability, overhead, and security of the new method. The results of a simulation and comparison to the original butterfly scheme are also presented.

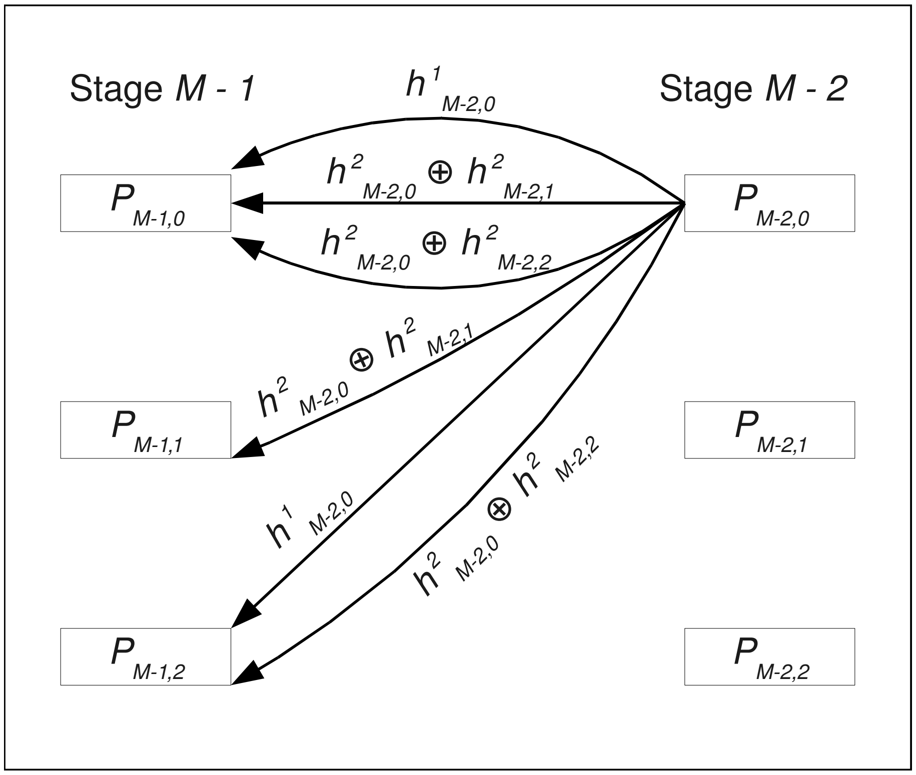
4.5.1 Verification Probability

As in the original butterfly scheme, it is assumed that the signature packet will be received correctly. To determine the probability of verifying a packet, consider the final stage, $M-1$. In this stage, if a packet is received correctly, then it can be verified. So, the probability of verifying any particular packet is simply $1-p$, where p is the packet error rate of the channel, assumed to be equal and independent for all packets.

For stage $M-2$, it is possible to calculate the probability of verifying the first or second half of a packet's hash, as follows. Figure 4.7 shows where the half-hashes for a given packet (packet $P_{M-2,0}$) are appended. For the purpose of calculating verification probability, $P_{M-2,0}$ can be any packet in stage $M-2$; the verification probability for all packets in a given stage will be the same, because of symmetry. $h^1_{M-2,0}$ is the first half of the hash of packet $P_{M-2,0}$, while $h^2_{M-2,0} \oplus h^2_{M-2,1}$ is the second half of the hash of packet $P_{M-2,0}$ XORed with the second half of the hash of packet $P_{M-2,1}$.

For completeness, Figure 4.8 also shows where the hash segments appended to a given packet (packet $P_{M-1,0}$) come from. However, in calculating the verification probability, it is easier to refer to Figure 4.7.

$\Lambda_{m,n}$ denotes the event that packet $P_{m,n}$ can be verified, while $\Lambda^1_{m,n}$ (or $\Lambda^2_{m,n}$) denotes the event that the first (or second) half of the hash of packet $P_{m,n}$ can be verified. $\Pi_{m,n}$ denotes the event that packet $P_{m,n}$ is received correctly.



Stage M - 1
$h^1_{M-2,0}$
Stage M - 2
$P_{M-1,0}$
$h^2_{M-2,0} \oplus h^2_{M-2,1}$
$P_{M-2,0}$
$h^2_{M-2,0} \oplus h^2_{M-2,2}$
$h^2_{M-2,0} \oplus h^2_{M-2,1}$
$P_{M-1,1}$
$P_{M-2,1}$
$h^2_{M-2,0} \oplus h^2_{M-2,2}$
$h^1_{M-2,0}$
$P_{M-1,2}$
$P_{M-2,2}$

Figure 4.7: Destinations of a single packet's half-hashes for the modified butterfly scheme.

For the first half of the hash of packet $P_{M-2,0}$:

$$P(\Lambda^1_{M-2,0} \mid \Pi_{M-2,0}) = P(\Lambda_{M-1,0} \cup \Lambda_{M-1,2} \mid \Pi_{M-2,0}) \tag{4.6}$$

That is, the probability of verifying packet $P_{M-2,0}$, given that it is received, is the probability that either packet $P_{M-1,0}$ or packet $P_{M-1,2}$ can be verified. Since $P_{M-1,0}$ and $P_{M-1,2}$ are both in stage $M-1$, the probability that they can be verified is the same as the probability that they are received. That is:

$$P(\Lambda^1_{M-2,0} \mid \Pi_{M-2,0}) = P(\Pi_{M-1,0} \cup \Pi_{M-1,2}) = 1 - p^2 \tag{4.7}$$

To verify the second half of the hash, at least one other packet, $P_{M-2,1}$ or $P_{M-2,2}$, is

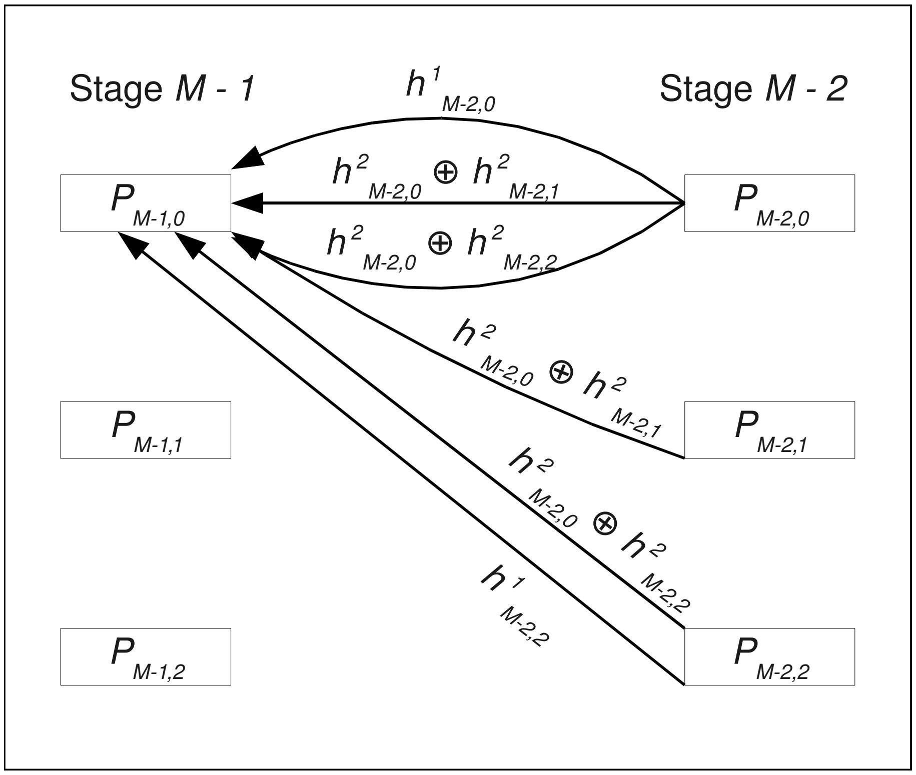

Stage M - 1
Stage M - 2
$h^1_{M-2,0}$
$h^2_{M-2,0} \oplus h^2_{M-2,1}$
$h^2_{M-2,0} \oplus h^2_{M-2,2}$
$P_{M-1,0}$
$P_{M-1,1}$
$P_{M-1,2}$
$P_{M-2,0}$
$P_{M-2,1}$
$P_{M-2,2}$
$h^2_{M-2,0} \oplus h^2_{M-2,1}$
$h^2_{M-2,0} \oplus h^2_{M-2,2}$
$h^1_{M-2,2}$

Figure 4.8: Sources of a single packet's hash segments for the modified butterfly scheme.

required from stage $M-2$. Conditioning on which packet is received yields:

$$\begin{aligned} P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) &= P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,1} \cap \Pi_{M-2,2} \cap \Pi_{M-2,0}) P(\Pi_{M-2,1} \cap \Pi_{M-2,2}) + \\ &\quad P(\Lambda^2_{M-2,0} \mid \overline{\Pi_{M-2,1}} \cap \Pi_{M-2,2} \cap \Pi_{M-2,0}) P(\overline{\Pi_{M-2,1}} \cap \Pi_{M-2,2}) + \\ &\quad P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,1} \cap \overline{\Pi_{M-2,2}} \cap \Pi_{M-2,0}) P(\Pi_{M-2,1} \cap \overline{\Pi_{M-2,2}}) \end{aligned} \tag{4.8}$$

From Figure 4.7, it can be seen that if both packets $P_{M-2,1}$ and $P_{M-2,2}$ have been received, then the second half of packet $P_{M-2,0}$ can be verified if any of packets $P_{M-1,0}$, $P_{M-1,1}$, or $P_{M-1,2}$ have been verified. If only packet $P_{M-2,2}$ has been received, then either $P_{M-1,0}$ or $P_{M-1,2}$ is needed; if only packet $P_{M-2,1}$ has been received, then either $P_{M-1,0}$ or $P_{M-1,1}$ is needed. Thus, Equation 4.8 becomes:

$$\begin{aligned}
P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) &= P(\Lambda_{M-1,0} \cup \Lambda_{M-1,1} \cup \Lambda_{M-1,2} \mid \Pi_{M-2,0})(1-p)^2 + \\
&\quad P(\Lambda_{M-1,0} \cup \Lambda_{M-1,2} \mid \Pi_{M-2,0})p(1-p) + \\
&\quad P(\Lambda_{M-1,0} \cup \Lambda_{M-1,1} \mid \Pi_{M-2,0})p(1-p) \\
&= P(\Pi_{M-1,0} \cup \Pi_{M-1,1} \cup \Pi_{M-1,2})(1-p)^2 + \\
&\quad P(\Pi_{M-1,0} \cup \Pi_{M-1,2})p(1-p) + P(\Pi_{M-1,0} \cup \Pi_{M-1,1})p(1-p) \\
&= (1-p^3)(1-p)^2 + (1-p^2)p(1-p) + (1-p^2)p(1-p) \\
P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) &= (1-p)^2(1+2p+2p^2-p^3) \qquad (4.9)
\end{aligned}$$

Similarly, it is possible to calculate the probability of verifying both halves of the hash of packet $P_{M-2,0}$ together. Again, this is done by conditioning on which of $P_{M-2,1}$ and $P_{M-2,2}$ is received. In this case, if both $P_{M-2,1}$ and $P_{M-2,2}$ have been received, or just $P_{M-2,2}$, then either $P_{M-1,0}$ or $P_{M-1,2}$ is needed to verify both halves of $P_{M-2,0}$. If only packet $P_{M-2,1}$ has been received, then either $P_{M-1,0}$, or both of $P_{M-1,1}$ and $P_{M-1,2}$, is needed.

$$\begin{aligned}
P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) &= P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \Pi_{M-2,1} \cap \Pi_{M-2,2} \cap \Pi_{M-2,0}) \times \\
&\quad P(\Pi_{M-2,1} \cap \Pi_{M-2,2}) + \\
&\quad P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \overline{\Pi_{M-2,1}} \cap \Pi_{M-2,2} \cap \Pi_{M-2,0}) \times \\
&\quad P(\overline{\Pi_{M-2,1}} \cap \Pi_{M-2,2}) + \\
&\quad P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \Pi_{M-2,1} \cap \overline{\Pi_{M-2,2}} \cap \Pi_{M-2,0}) \times \\
&\quad P(\Pi_{M-2,1} \cap \overline{\Pi_{M-2,2}}) \\
&= P(\Pi_{M-1,0} \cup \Pi_{M-1,2})(1-p)^2 + \\
&\quad P(\Pi_{M-1,0} \cup \Pi_{M-1,2})p(1-p) +
\end{aligned}$$

$$\begin{aligned}
& P(\Pi_{M-1,0} \cup [\Pi_{M-1,1} \cap \Pi_{M-1,2}])p(1-p) \\
&= (1-p^2)(1-p)^2 + (1-p^2)p(1-p) + \\
& [P(\Pi_{M-1,0}) + P(\Pi_{M-1,1} \cap \Pi_{M-1,2}) - \\
& P(\Pi_{M-1,0} \cap \Pi_{M-1,1} \cap \Pi_{M-1,2})]p(1-p) \\
&= (1-p^2)(1-p)^2 + p(1-p)(1-p^2) + \\
& [1-p+(1-p)^2-(1-p)^3]p(1-p) \\
P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) &= (1-p)^2(1+2p+p^2-p^3) \qquad (4.10)
\end{aligned}$$

Ω_m is defined to be the probability of verifying a packet (either the first or second half) in stage m. It is possible to calculate Ω_{M-2} using Equations 4.7, 4.9, and 4.10, as follows:

$$\begin{aligned}
\Omega_{M-2} &= P(\Lambda^1_{M-2,0} \cup \Lambda^2_{M-2,0}) = P(\Lambda^1_{M-2,0} \cup \Lambda^2_{M-2,0} \mid \Pi_{M-2,0})P(\Pi_{M-2,0}) \\
&= [P(\Lambda^1_{M-2,0} \mid \Pi_{M-2,0}) + P(\Lambda^2_{M-2,0} \mid \Pi_{M-2,0}) - \\
& P(\Lambda^1_{M-2,0} \cap \Lambda^2_{M-2,0} \mid \Pi_{M-2,0})]P(\Pi_{M-2,0}) \\
&= [(1-p^2) + (1-p)^2(1+2p+2p^2-p^3) - (1-p)^2(1+2p+p^2-p^3)](1-p) \\
\Omega_{M-2} &= (1-p)^2(1+p+p^2-p^3) \qquad (4.11)
\end{aligned}$$

It is now possible to calculate the verification probability of stage $M-3$ (and subsequent stages) using Ω_{M-2}. The difference is that the verification probability is used in place of the probability of receiving a packet. From the point of view of the packets in stage $M-3$, not verifying a packet from stage $M-2$ is equivalent to not receiving that packet, so this substitution makes sense. For example, to calculate the verification probability for the first half of packet $P_{M-3,0}$:

$$P(\Lambda^1_{M-3,0} \mid \Pi_{M-3,0}) = P(\Lambda_{M-2,0} \cup \Lambda_{M-2,2}) = 1-(1-\Omega_{M-2})^2 \qquad (4.12)$$

In other words, the verification probability for the first half of $P_{M-3,0}$ is the probability that either $P_{M-2,0}$ or $P_{M-2,2}$ can be verified. A simplifying assumption is made here; namely, that the verification probability of $P_{M-2,0}$ is independent of the verification probability of $P_{M-2,2}$. This is not strictly true, since verifying $P_{M-2,0}$ depends on some of the same packets used to verify $P_{M-2,2}$. However, it is a reasonably good approximation, as will be seen in Section 4.5.8.

Using the same simplifying assumption, for the second half of $P_{M-3,0}$:

$$\begin{aligned}
P(\Lambda^2_{M-3,0} \mid \Pi_{M-3,0}) &= P(\Lambda^2_{M-3,0} \mid \Pi_{M-3,1} \cap \Pi_{M-3,2} \cap \Pi_{M-3,0}) P(\Pi_{M-3,1} \cap \Pi_{M-3,2}) + \\
&\quad P(\Lambda^2_{M-3,0} \mid \overline{\Pi_{M-3,1}} \cap \Pi_{M-3,2} \cap \Pi_{M-3,0}) P(\overline{\Pi_{M-3,1}} \cap \Pi_{M-3,2}) + \\
&\quad P(\Lambda^2_{M-3,0} \mid \Pi_{M-3,1} \cap \overline{\Pi_{M-3,2}} \cap \Pi_{M-3,0}) P(\Pi_{M-3,1} \cap \overline{\Pi_{M-3,2}}) \\
&= P(\Lambda_{M-2,0} \cup \Lambda_{M-2,1} \cup \Lambda_{M-2,2})(1-p)^2 + \\
&\quad P(\Lambda_{M-2,0} \cup \Lambda_{M-2,2}) p(1-p) + P(\Lambda_{M-2,0} \cup \Lambda_{M-2,1}) p(1-p) \\
&= (1 - [1 - \Omega_{M-2}]^3)(1-p)^2 + 2(1 - [1 - \Omega_{M-2}]^2) p(1-p) \\
P(\Lambda^2_{M-3,0} \mid \Pi_{M-3,0}) &= \Omega_{M-2}(1-p)(3 - 3\Omega_{M-2} + \Omega^2_{M-2} + p + \Omega_{M-2}p - \Omega^2_{M-2}p)
\end{aligned} \tag{4.13}$$

For both halves of $P_{M-3,0}$ together:

$$\begin{aligned}
P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \Pi_{M-3,0}) &= P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \Pi_{M-3,1} \cap \Pi_{M-3,2} \cap \Pi_{M-3,0}) \times \\
&\quad P(\Pi_{M-3,1} \cap \Pi_{M-3,2}) + \\
&\quad P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \overline{\Pi_{M-3,1}} \cap \Pi_{M-3,2} \cap \Pi_{M-3,0}) \times \\
&\quad P(\overline{\Pi_{M-3,1}} \cap \Pi_{M-3,2}) + \\
&\quad P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \Pi_{M-3,1} \cap \overline{\Pi_{M-3,2}} \cap \Pi_{M-3,0}) \times \\
&\quad P(\Pi_{M-3,1} \cap \overline{\Pi_{M-3,2}})
\end{aligned}$$

$$\begin{aligned}
&= P(\Lambda_{M-2,0} \cup \Lambda_{M-2,2})(1-p)^2 + \\
&\quad P(\Lambda_{M-2,0} \cup \Lambda_{M-2,2})p(1-p) + \\
&\quad P(\Lambda_{M-2,0} \cup [\overline{\Lambda_{M-2,0}} \cap \Lambda_{M-2,1} \cap \Lambda_{M-2,2}])p(1-p) \\
&= (1-[1-\Omega_{M-2}]^2)(1-p)^2 + (1-[1-\Omega_{M-2}]^2)p(1-p) + \\
&\quad (\Omega_{M-2} + \Omega_{M-2}^2[1-\Omega_{M-2}])p(1-p) \\
P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \Pi_{M-3,0}) &= \Omega_{M-2}(1-p)(2-\Omega_{M-2}+p+\Omega_{M-2}p-\Omega_{M-2}^2 p)
\end{aligned} \tag{4.14}$$

Finally, the verification probability for a packet in stage $M-3$ can be calculated using Equations 4.12, 4.13, and 4.14:

$$\begin{aligned}
\Omega_{M-3} &= P(\Lambda^1_{M-3,0} \cup \Lambda^2_{M-3,0} \mid \Pi_{M-3,0})P(\Pi_{M-3,0}) \\
&= [P(\Lambda^1_{M-3,0} \mid \Pi_{M-3,0}) + P(\Lambda^2_{M-3,0} \mid \Pi_{M-3,0}) - \\
&\quad P(\Lambda^1_{M-3,0} \cap \Lambda^2_{M-3,0} \mid \Pi_{M-3,0})]P(\Pi_{M-3,0}) \\
&= [1-(1-\Omega_{M-2})^2 + \Omega_{M-2}(1-p)(3-3\Omega_{M-2}+\Omega_{M-2}^2+p+\Omega_{M-2}p-\Omega_{M-2}^2 p) - \\
&\quad \Omega_{M-2}(1-p)(2-\Omega_{M-2}+p+\Omega_{M-2}p-\Omega_{M-2}^2 p)](1-p) \\
\Omega_{M-3} &= [1+(1-\Omega_{M-2})^2(\Omega_{M-2}[1-p]-1)](1-p)
\end{aligned} \tag{4.15}$$

Generalizing Equation 4.15 for $m < M-1$ gives:

$$\Omega_m = [1+(1-\Omega_{m+1})^2(\Omega_{m+1}[1-p]-1)](1-p) \tag{4.16}$$

and:

$$\Omega_{M-1} = 1-p \tag{4.17}$$

Unfortunately, there is no convenient closed-form expression for Equation 4.16. However, by setting $\Omega_m = \Omega_{m+1}$, it is possible to find the value to which it converges.

$$\Omega_{min} = \frac{3 - 2p \pm \sqrt{4p+1}}{2(1-p)} \tag{4.18}$$

This is useful as an approximation to the verification probability for a given value of p, regardless of the number of stages used.

4.5.2 Verification Probability for Both Halves

It is also possible to derive the verification probability under the assumption that both halves of a packet must be verified in order to use that packet for further verification. The math is the same as above, except that the verification probability Ω_m is now defined as the probability of verifying both halves of a packet in stage m. That is,

$$\Omega_m = P(\Lambda^1_{m,0} \cap \Lambda^2_{m,0}) = P(\Lambda^1_{m,0} \cap \Lambda^2_{m,0} \mid \Pi_{m,0})P(\Pi_{m,0}) \tag{4.19}$$

Note that any packet in stage m could be used in place of $P_{m,0}$. Redoing the math with this new definition yields:

$$\Omega_m = \Omega_{m+1}(1-p)(2 - \Omega_{m+1} + p + \Omega_{m+1}p - \Omega^2_{m+1}p) \tag{4.20}$$

with Ω_{M-1} the same as in Equation 4.17.

4.5.3 Overhead

The average overhead per packet is calculated as follows. There are 2 full hashes (4 halves) appended to each of the T packets in the block, except for the packets in the first stage, of which there are N. There is also one signature packet. Therefore, the average overhead per packet is:

$$O_{avg} = \frac{2s_h(T-N)+s_s}{T} = 2s_h - \frac{2s_h}{M} + \frac{Ns_h}{T} = 2s_h - \frac{s_h}{M} \tag{4.21}$$

where s_s is the size of the signature packet and s_h is the size of a hash. This is very close to the overhead of the original butterfly scheme, differing only in the definitions of T, M, and N. For a 160-bit hash, with $M = 5$, the overhead of the modified scheme works out to 288 bits, or 36 bytes, per packet.

4.5.4 Time and Space Complexity

The operations that take the most time are the hashing and signing of packets; compared to these, the XOR and other operations like copying take a negligible amount of time. For each block of T packets, it is necessary to compute T hashes and one signature. The amount of computation time needed for one block is therefore $Tt_h + t_s$, where t_s is the time to sign a packet and t_h is the time to compute a hash.

In terms of storage space, it is necessary to buffer the hashes for an entire stage of packets. Once the hashes have been appended to the next stage of packets, they can be discarded. Therefore the space needed at the sender is Ns_h.

At the receiver, none of the packets can be verified until the signature packet is received, so it is necessary to buffer the entire block. The space needed at the receiver is therefore Ts_h.

4.5.5 Sender and Receiver Delay

At the sender, it is necessary to buffer a packet long enough to calculate the hash segments that are to be appended to it. This will depend on the amount of time between packets. If the time between packets is large enough, then all the hash segments for stage M should have been generated by the time the first packet of stage $M+1$ is ready to send. In this case, it can be sent immediately, with no delay. If, however, the time between packets is small, then it may be necessary to calculate up to N hashes before sending a given packet. In this case, the sender delay is Nt_h.

At the receiver, none of the packets can be verified until the signature packet has been received. So, the maximum possible delay (for authentication of a packet, not reception) at the receiver is of order T. The exact amount of the delay will depend on how long it takes to send a packet across the network.

4.5.6 Security Concerns

One concern with the modified butterfly method is the reduction in security that comes from dividing each hash into two halves. One might think that an attacker could successfully forge a valid packet-hash pair with only half the work required for a complete hash. However, the chaining of the hashes mitigates the attacker's advantage somewhat. It is not enough to simply take a hash and find the corresponding packet by brute force; an attacker would have to find a valid packet that also contains the hashes of the previous packets. Therefore, although there is a reduction in security, it should not be reduced as far as half of the number of bits of the hash. A more thorough analysis would be necessary in order to have complete confidence in this scheme; however, for the purposes of this thesis, it suffices to say that the security of this method appears to be adequate for the proposed usage.

4.5.7 Simulation

A Matlab simulation was written to compare the modified butterfly scheme to the original. The Matlab code is shown in Appendix A, with comments.

The packet hashes were simulated using random numbers. The program was designed to iterate through several error rates, with a predefined number of simulation runs at each error rate. For each run of the simulation, the random hashes were generated, and the local kernel matrices were calculated for each packet and used to build the hash chains. Once that was done, errors were randomly introduced into the packets at the current error rate. The received packets were then verified, and a count was kept of how many were verified by the first half-hash, the second half-hash, both halves, or either half. These counts were accumulated over all the simulation runs, and the totals were stored for each error rate. The number of correctly-received packets was also accumulated over all the simulation runs, and was used to calculate the verification probability at each error rate.

4.5.8 Simulation Results

The modified scheme was simulated 3000 times, and the results were plotted against the packet error rate, p, for $T = 1280, N = 256, M = 5$. The packet error rate varied from 5% to 50%. Zhang *et al*'s butterfly scheme was also simulated for $T = 1024, N = 128, M = 8$. Since the two schemes have differing block sizes, it was not possible to use the same block size, but these are close enough for a fair comparison. The results are shown in Fig. 4.9. Shown in the graph are the average verification probabilities for the first half, second half, both halves, and either half of a packet in the modified scheme; the average verification probability of a packet in the original butterfly scheme; and the verification probability for the modified scheme that was calculated using Equation 4.16.

As seen from the graph, the modified butterfly scheme is able to authenticate a packet, at least partially, with a much higher probability than the original butterfly method. In addition, the modified scheme can completely authenticate a packet with a probability very close to the original scheme, even outperforming the original at a high enough error rate. The 95% confidence intervals for the verification probabilities are shown in Table 4.3.

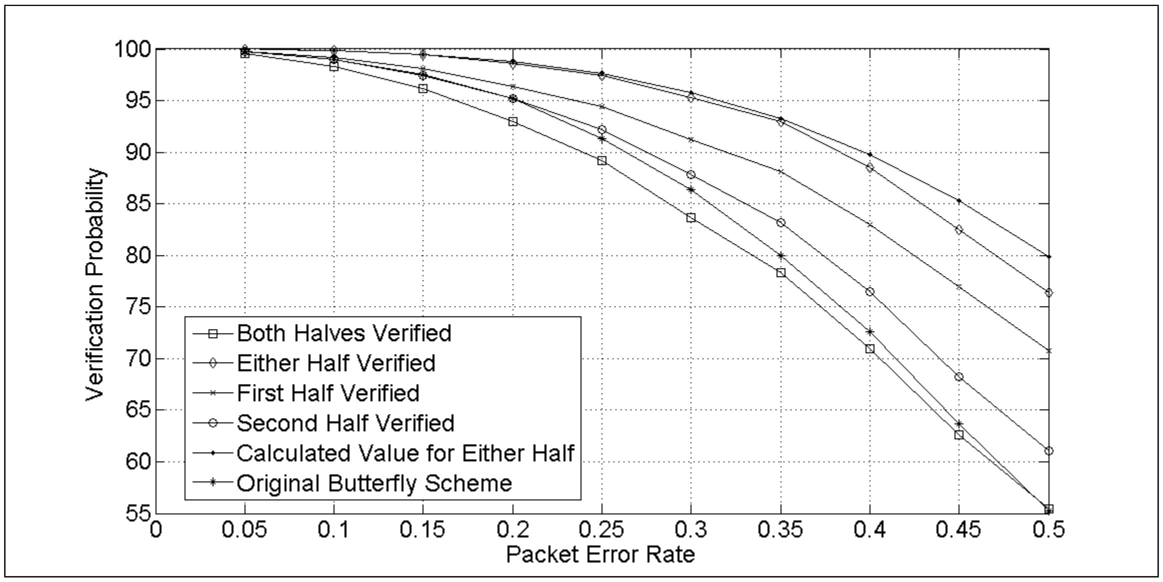


Figure 4.9: Simulation results for the modified butterfly scheme with partial authentication of packets.

Table 4.3: 95% confidence intervals for simulation results for the modified butterfly scheme.

Error Rate	**First Half**	**Second Half**	**Both Halves**	**Either Half**
0.0500	±0.4068	±0.3346	±0.4375	±0.0613
0.1000	±0.7717	±0.9344	±1.3200	±0.2642
0.1500	±1.2762	±1.2691	±1.6342	±0.5322
0.2000	±1.3013	±1.8704	±2.2415	±0.7189
0.2500	±1.7375	±2.9066	±3.2474	±1.2879
0.3000	±3.3167	±3.8325	±4.2543	±2.5673
0.3500	±3.1592	±3.9112	±3.7059	±2.5589
0.4000	±4.0120	±5.0492	±5.3675	±3.7541
0.4500	±6.8171	±6.9660	±7.6072	±6.0436
0.5000	±7.5213	±7.7284	±7.5490	±7.7155

At higher error rates, there is some deviation between the calculated verification probability and the simulated results. This is due to the simplifying assumption that the verification probabilities of packets in the same stage are independent. While not strictly true, the results show that this assumption yields a fairly good approximation (within five percent, even at the highest simulated error rate). It should also be noted that a channel with a 50% error rate would be virtually unusuable, and that for all practical error rates, the calculated value is very close to the simulated one.

In the simulation above, it was assumed that partially verified packets could still be used to authenticate other packets. A simulation was also performed assuming that a packet must be completely verified in order to be able to authenticate other packets. The results of this simulation are shown in Fig. 4.10. Once again, the modified scheme outperforms the original in partial authentication of packets, although not by as large a margin. The modified scheme is consistently lower than the original in complete authentication of packets, except at very high error rates (not pictured).

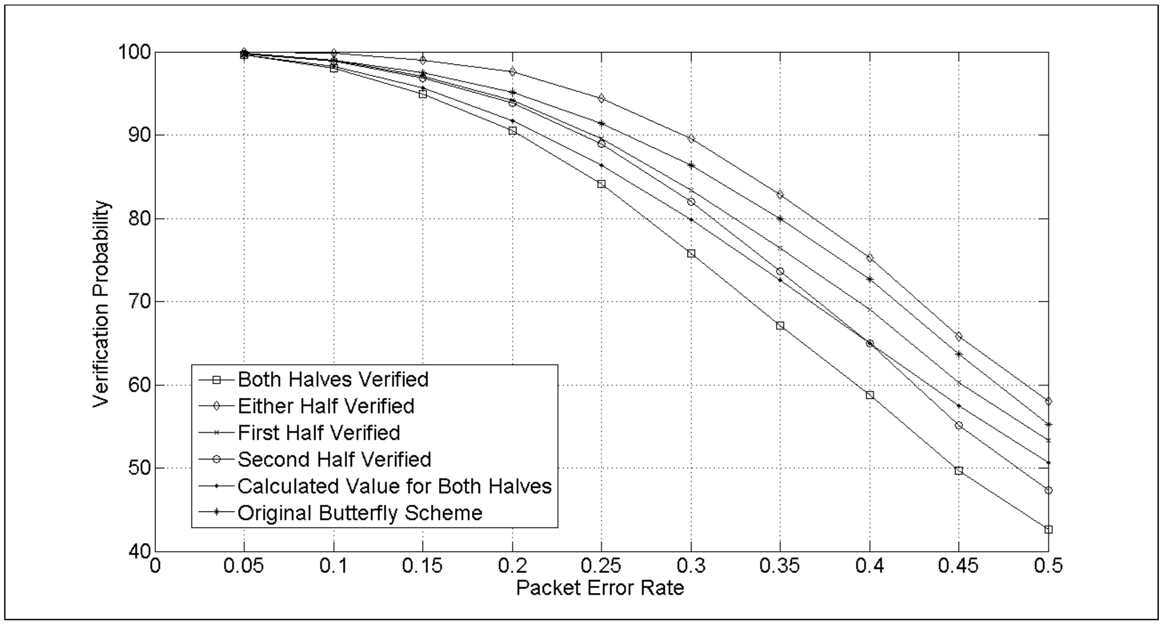


Figure 4.10: Simulation results for the modified butterfly scheme with complete authentication of packets.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, a categorization of authentication schemes was presented, expanding the classification given by Menezes *et al* in [1]. This categorization was based on a number of factors, mainly the type of authentication (user or data), the way in which authentication information is transmitted (embedded or appendix), and the secrecy of the authentication information (covert or overt).

In addition, a novel method of achieving data authentication of a real-time audio stream was presented. This method is an extension of previous work that has been done in the area of hash-chaining authentication schemes. The new method, the modified butterfly scheme, divides and mixes the hashes of each packet in order to spread the information among a larger number of hash chains. The modified scheme adds the potential for a packet to be partially authenticated, even if it cannot be fully authenticated. Theoretical and simulated results were presented, showing that the modified butterfly scheme outperforms the original butterfly scheme using the same amount of overhead.

5.2 Future Work

There are several avenues of further work that can be done on the modified butterfly hash-chaining scheme. A more rigorous security analysis, describing the degrees of freedom given to a potential attacker, is necessary in order to have complete confidence in the security of the system. A different way of combining the half-hashes, rather than XORing them, could provide better security and performance. In addition, it may be possible to improve the performance of the scheme by dividing each packet's hash into more than two parts. It should be determined how best to divide the hashes, in order to balance security and performance. Finally, this scheme has potential applications in unequal authentication; that is, increasing the verification probability of certain bits at the expense of others. This could be done by adding some redundancy to the packets before hashing them. Again, this would need to be balanced against the security requirements of the system.

Appendix A

Matlab Simulation Code

modified_butterfly.m

```
clear all;

% vector to store final verification probabilities
RSLT = [];
% vector to store final verification probabilities, by stage
ERR_RSLT = [];
% vector to store confidence intervals
ci = [];
RSLT2 = [];

% mode = 1 modified scheme
% mode = 2 original butterfly scheme
mode = 1;

% number of stages, M
M=4;
```

```
if (mode == 1)
    % total number of packets, T
    T = (M+1)*2^(2*M);
    % number of packets per stage, N
    pkts_in_stage = 2^(2*M);
    % number of hash segments appended to each packet
    max_col_index = 4;
    % hashes divided into this many segments
    hash_count = 2;
elseif (mode == 2)
    T=(M+1)*2^M;
    pkts_in_stage = 2^(M);
    max_col_index = 2;
    hash_count = 1;
end % if


% packet error rate
for Err_rate = 0.05:0.05:0.50
    % print error rate
    Err_rate

    % variables to track the number of packets verified
    % cumulative over all simulation runs
    HH=0; % first half verified
    HL=0; % second half verified
    HT=0; % both halves verified
    HE=0; % either half verified

    HH_all=[]; % first half verified
    HL_all=[]; % second half verified
    HT_all=[]; % both halves verified
```

```
HE_all=[]; % either half verified

% vectors to track the number of packets verified in each stage
% cumulative over all simulation runs
lst_stg_ver_L =  zeros(M,1); % second half verified
lst_stg_ver_H =  zeros(M,1); % first half verified
lst_stg_ver_B =  zeros(M,1); % both halves verified
lst_stg_ver_E =  zeros(M,1); % either half verified

% total number of packets received correctly
% cumulative over all simulation runs
pkt_count = 0;

% number of simulation runs
for(Iterate = 1:3000)
    % generate packets
    Orig_pkt_set = zeros(T,hash_count+max_col_index+1);
    Rec_pkt_set = Orig_pkt_set;
    H_ver = zeros(T,hash_count);

    % generate hash of each packet (represented by random numbers)
    Orig_pkt_set(:,2:2+hash_count-1) = randint(T,hash_count,[1 99999]);

    % build the hash chain
    for i=0:M-1
        H = (Orig_pkt_set(i*pkts_in_stage+1:(i+1)*pkts_in_stage,2:2+hash_count-1))';
        for j=1:pkts_in_stage
            K=calc_kernel(i,j-1,pkts_in_stage,max_col_index,hash_count,mode);
            Orig_pkt_set((i+1)*pkts_in_stage+j, ...
                1+hash_count+1:1+hash_count+1+max_col_index-1) = H(:)'*K;
        end % for j
    end % for i
```

```
Rec_pkt_set = Orig_pkt_set;

% generate errors in the received packets, according to error rate
m=rand(T,1);
err_index = find(m < Err_rate);
Rec_pkt_set(err_index,1:hash_count+max_col_index+1) = ...
    -1000*rand(numel(err_index),hash_count+max_col_index+1);

% do the verification, starting from the last row
ver_index  = find(Rec_pkt_set(M*pkts_in_stage+1:(M+1)*pkts_in_stage,1)>=0);

H_ver(M*pkts_in_stage+ver_index,1:hash_count) = ...
    ~H_ver(M*pkts_in_stage+ver_index,1:hash_count);

% work backwards to verify the packets in the remaining stages
for i=M-1:-1:0
    H2= (Rec_pkt_set(i*pkts_in_stage+1:(i+1)*pkts_in_stage,2:2+hash_count-1))';
    for j=1:pkts_in_stage
        if ( (Rec_pkt_set((i+1)*pkts_in_stage+j,1) >= 0) ...
            && (sum(H_ver((i+1)*pkts_in_stage+j,:)) >= 1) )
            K=calc_kernel(i,j-1,pkts_in_stage,max_col_index,hash_count,mode);
            Hash_chain = H2(:)'*K;
            %check each entry by its corresponding member in the chain
            for (k=1:max_col_index)
                pkts_involved = find(K(:,k) == 1);
                if (Hash_chain(k) ==  ...
                    Rec_pkt_set((i+1)*pkts_in_stage+j,hash_count+1+k))
                    for t = 1:numel(pkts_involved)
                        H_ver(i*pkts_in_stage + ...
                            ceil(pkts_involved(t)/hash_count), ...
                            hash_count-mod(pkts_involved(t),hash_count)) = 1;
                    end % for t
                end % if
            end % for k
        end % if
    end % for j
end % for i

% accumulate the number of verified packets
HH = HH+numel(find(H_ver(:,1)==1));
HL = HL+numel(find(H_ver(:,2)==1));
HT = HT+numel(find(sum(H_ver') == 2));
HE = HE+numel(find(sum(H_ver') > 0));

% track overall verification probability for each run
% used to calculate confidence interval
HH_all = [HH_all; 100*numel(find(H_ver(:,1)==1))/(T-numel(err_index))];
HL_all = [HL_all; 100*numel(find(H_ver(:,2)==1))/(T-numel(err_index))];
HT_all = [HT_all; 100*numel(find(sum(H_ver') == 2))/(T-numel(err_index))];
HE_all = [HE_all; 100*numel(find(sum(H_ver') > 0))/(T-numel(err_index))];

% accumulate the total number of packets correctly received
pkt_count = pkt_count+T-numel(err_index);

% accumulate the number of verified packets by stage
for mm =1:M
    lst_stg_ver_H(mm) = lst_stg_ver_H(mm) + ...
        numel(find(H_ver(mm*pkts_in_stage+1:(mm+1)*pkts_in_stage,1) == 1));
    lst_stg_ver_L(mm) = lst_stg_ver_L(mm) + ...
        numel(find(H_ver(mm*pkts_in_stage+1:(mm+1)*pkts_in_stage,2) == 1));
    lst_stg_ver_B(mm) = lst_stg_ver_B(mm) + ...
        numel(find(sum(H_ver(mm*pkts_in_stage+1:(mm+1)*pkts_in_stage,:)') ...
        == 2));
```

```
            lst_stg_ver_E(mm) = lst_stg_ver_E(mm) + ...
                numel(find(sum(H_ver(mm*pkts_in_stage+1:(mm+1)*pkts_in_stage,:)') ...
                >= 1));
        end % for mm
    end  % for Iterate

    % print the verification probabilities for the current error rate
    % relative numbers - percentage of correctly-received packets
    [100*HH/pkt_count 100*HL/pkt_count 100*HE/pkt_count 100*HT/pkt_count]

    % add verification probabilities for current error rate to results table
    % relative numbers - percentage of correctly-received packets
    tmp =   [Err_rate 100*HH/pkt_count 100*HL/pkt_count 100*HE/pkt_count ...
            100*HT/pkt_count];
    RSLT = [RSLT;tmp];

    RSLT2 = [RSLT2; Err_rate mean(HH_all) mean(HL_all) mean(HE_all) mean(HT_all)];

    % add confidence intervals for current error rate to results table
    ci =    [ci; [Err_rate 1.96*std(HH_all) 1.96*std(HL_all) 1.96*std(HE_all) ...
            1.96*std(HT_all)] ];

    % add verification probabilities by stage for current error rate to results table
    % absolute numbers - percentage of all packets sent
    tmp = [ [Err_rate; Err_rate; Err_rate; Err_rate] ...
            lst_stg_ver_H./(Iterate*pkts_in_stage) ...
            lst_stg_ver_L./(Iterate*pkts_in_stage) ...
            lst_stg_ver_B./(Iterate*pkts_in_stage)  ...
            lst_stg_ver_E./(Iterate*pkts_in_stage)];
    ERR_RSLT = [ERR_RSLT; tmp];

end % for Err_rate
```

calc_kernel.m

```
% calculate the local kernel matrix for a given packet
function[K] = calc_kernel(m,n,total_pkts_in_stage,max_col_index,hash_count,mode)

    % m : stage index
    % n : index of packet in stage
    % other variables are as defined in modified_butterfly.m

    if (mode == 1)
        bit_size = ceil(log2(total_pkts_in_stage));
        bit_map = de2bi(n,bit_size);

        % initialize matrix to zeroes
        K = zeros(hash_count*total_pkts_in_stage,max_col_index);

        % iterate through columns of the matrix
        for j=1:max_col_index
            bit_map2 = bit_map;

            % apply equations for modified butterfly scheme
            switch (j)
                case 1
                    K(2*n+1,j) = 1;
                case 2
                    K(2*n+2,j) = 1;
                    bit_map2(2*m+1) = ~bit_map2(2*m+1);
                    K(2*bi2de(bit_map2)+2,j) = 1;
                case 3
                    bit_map2(2*m+2) = ~bit_map2(2*m+2);
                    K(2*bi2de(bit_map2)+1,j) = 1;
                case 4
                    K(2*n+2,j) = 1;
                    bit_map2(2*m+2) = ~bit_map2(2*m+2);
                    K(2*bi2de(bit_map2)+2,j) = 1;
            end % switch
        end % for j
    elseif (mode == 2)
        bit_size = ceil(log2(total_pkts_in_stage));
        bit_map = de2bi(n,bit_size);

        % initialize matrix to zeroes
        K = zeros(hash_count*total_pkts_in_stage,max_col_index);

        % iterate through columns of the matrix
        for j=1:max_col_index
            bit_map2 = bit_map;

            % apply equations for original butterfly scheme
            switch (j)
                case 1
                    % direct link
                    K(hash_count*n+1,j) = 1;
                case 2
                    % flip bit corresponding to current stage
                    bit_map2(m+1) = ~bit_map2(m+1);
                    K(bi2de(bit_map2)+hash_count,j) = 1;
            end % switch
        end % for j
    end % if
```

err_calc.m

```
% given packet error rate and number of stages,
% returns the verification probability at each stage
function [Omega] = err_calc(p,Total_Stage);

    % initialize all to zero
    Omega = zeros(Total_Stage,1);

    % formulas for last and second-last stage
    Omega(Total_Stage) = 1-p;
    Omega(Total_Stage-1) = (1-2*p^3+p^4)*(1-p);

    % calculate other stages iteratively
    for mm=Total_Stage-2:-1:1
     Omega(mm) = ( 1+(1-Omega(mm+1))^2 *(Omega(mm+1)*(1-p)-1) )*(1-p);
    end
```

References

[1] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* Boca Raton: CRC Press, 1997.

[2] S. Katzenbeisser and F. Petitcolas, *Information Hiding Techniques for Steganography and Digital Watermarking.* Norwood, MA, USA: Artech House, Inc., 2000.

[3] National Institute of Standards and Technology, *Federal Information Processing Standards Publication 180-3: Secure Hash Standard*, October 2008.

[4] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[5] National Institute of Standards and Technology, *Federal Information Processing Standards Publication 186-3: Digital Signature Standard*, June 2009.

[6] M. Robertson, "Establishing confidence level measurements for remote user authentication in privacy-critical systems," Master's thesis, University of Waterloo, Waterloo, Ont., 2009.

[7] F. Bergadano, D. Gunetti, and C. Picardi, "User authentication through keystroke dynamics," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 4, pp. 367–397, 2002.

[8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[9] M. Wiener, "Cryptanalysis of short RSA secret exponents," *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 553–558, 1990.

[10] J. Pollard, "Theorems on factorization and primality testing," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 76, pp. 521–528, Cambridge University Press, 1974.

[11] K. Nyberg and R. A. Rueppel, "A new signature scheme based on the DSA giving message recovery," in *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security*, (New York, NY, USA), pp. 58–61, ACM, 1993.

[12] K. Nyberg and R. A. Rueppel, "Message recovery for signature schemes based on the discrete logarithm problem," *Designs, Codes and Cryptography*, vol. 7, no. 1, pp. 61–81, 1996.

[13] T. Clancy, *Patriot Games.* New York: Berkley Books, 1988.

[14] J. Eggers and B. Girod, *Informed Watermarking.* Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[15] M. Schneider and S. Chang, "A robust content based digital signature for image authentication," in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, pp. 227–230, 1996.

[16] R. Gennaro and P. Rohatgi, "How to sign digital streams," *Inf. Comput.*, vol. 165, no. 1, pp. 100–116, 2001.

[17] A. Perrig, R. Canetti, D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *IEEE Symposium on Security and Privacy*, pp. 56–75, 2000.

[18] P. Golle and N. Modadugu, "Authenticating streamed data in the presence of random packet loss," *Proc. Network and Distributed System Security Symp.*, pp. 13–22, Feb. 2001.

[19] C. Wong and S. Lam, "Digital signatures for flows and multicasts," *IEEE/ACM transactions on Networking*, vol. 7, no. 4, pp. 502–513, 1999.

[20] Z. Zhang, Q. Sun, and W. Wong, "A proposal of butterfly-graph based stream authentication over lossy networks," in *Proc. IEEE International Conference on Multimedia & Expo*, 2005.