

國立交通大學

資訊工程學系

碩士論文

有效率抵抗污染攻擊的多點傳送認證協定

Lightweight, Pollution-Attack Resistant Multicast

Authentication Scheme



研究生：林亞正

指導教授：謝續平 博士

中華民國九十四年六月

有效率抵抗污染攻擊的多點傳送認證協定

Lightweight, Pollution-Attack Resistant Multicast
Authentication Scheme

研究生：林亞正

Student: Ya-Jeng Lin

指導教授：謝續平 博士

Advisor: Dr. Shiuh-Pyng Shieh

國立交通大學
資訊工程學系
碩士論文



A Thesis
Submitted to
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science and Information Engineering

June 2004
Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

有效率抵抗污染攻擊的多點傳送認證協定

研究生：林亞正

指導教授：謝續平 博士

國立交通大學資訊工程學系

摘要

在每個多點傳送的封包加上數位簽章會耗費很多資源。為了減少安全性所帶來的花費，有人提出平攤產生簽章花費的方法。另一方面，為了容忍在多點傳送環境中封包遺失的情況，通常會加上容錯的編碼方法。然而當攻擊者傳送不合法封包時，容錯的解碼流程會被搗亂，因而產生錯誤的資訊。這種攻擊方式被稱之為污染攻擊。目前解決污染攻擊的方法非常沒有效率，為了改善這個問題，我們提出了一個有效率解決污染攻擊的多點傳送認證協定。



Lightweight, Pollution-Attack Resistant Multicast

Authentication Scheme

Student: Ya-Jeng Lin

Advisor: Shih-Pyng Shieh

Department of Computer Science and Information Engineering

National Chaio Tung University

Abstract

Signing every multicast packet incurs high overhead. To reduce security overhead, signature amortization is proposed. On the other hand, to tolerate packet loss, erasure codes is used to enhance signature amortization. However, signature amortization is weak against pollution attack which occurs when an attacker injects packets to disturb the erasure decoding procedure, and consequently denies the service. Current solution to this pollution attack is heavyweight and inefficient. To cope with this problem, we propose a new lightweight, pollution-attack resistant multicast authentication scheme (PARM), which generates evidence for each packet that can validated by the receiver. This approach effectively resists pollution attacks and has better performance than previous proposed solutions.

誌 謝

首先感謝交通大學網路安全實驗室在我兩年研究生生活中給了我許多寶貴的經驗以及知識，感謝在這兩年中，指導老師謝續平教授給我的諄諄教導，教導我如何做研究，同時也感謝實驗室的學長、同學以及學弟們的互相討論、切磋，讓我的研究可以更完整。另外父母的全力支持讓我能無後顧之憂的專心做研究，非常感謝他們。

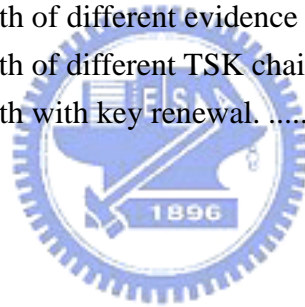


Table of contents

1.	Introduction.....	1
2.	Related work	5
2.1.	Signature amortization	5
2.2.	SAIDA	8
2.3.	Distillation codes	10
3.	Proposed scheme.....	13
3.1.	Lightweight and pollution attack resistant multicast authentication protocol (PARM).....	13
3.1.1.	Initialization Phase.....	13
3.1.2.	Evidence Generation Phase.....	15
3.1.3.	Evidence Validation Phase	17
3.1.4.	Temporal key renewal phase.....	19
3.2.	Practical considerations	21
3.3.	Attacks resistance.....	22
4.	Comparison	24
4.1.	Storage overhead.....	24
4.2.	Communication overhead	25
4.3.	Computation overhead	25
5.	Security analysis	28
6.	Evaluation	30
7.	Conclusion	34
	References.....	35

List of Figures

Figure 2-1 Merkle Hash Tree.....	6
Figure 2-2 SAIDA (Signature Amortization using the Information Dispersal Algorithm).....	9
Figure 2-3 Distillation codes.....	11
Figure 3-1 Temporal key pair generation.....	14
Figure 3-2 Appearance times table.	16
Figure 3-3 Evidence generation phase.....	17
Figure 3-4 Evidence validation phase.....	18
Figure 3-5 Temporal key renewal phase.....	21
Figure 4-1 Computation overhead at sender.	27
Figure 4-2 Computation overhead at receiver.....	27
Figure 6-1 The security strength of different evidence size (1).....	30
Figure 6-2 The security strength of different evidence size (2).....	31
Figure 6-3 The security strength of different TSK chain length.....	32
Figure 6-4 The security strength with key renewal.	32



1. Introduction

Multicast protocol enables a sender to efficiently disseminate digital media data to many receivers. Due to the time-sensitive requirement of some applications, reliable transmission protocol like TCP (Transmission Control Protocol) is impractical for multicast. Therefore, unreliable transmission protocol such as UDP (User Datagram Protocol) is generally adopted for multicast applications. Multicast protocol is suitable for many applications, for example, video transmission, live broadcast, stock quotation or news feed. These applications have the common characteristics that the receiver might be plenty and the communication data is time-sensitive. However security is an important issue for multicast to ensure secure communication between sender and receivers. An attacker is able to impersonate sender to send malicious packets to receivers and the malicious information might injure the receivers or intercept the communication. To defend against forged packets injected by attackers, multicast authentication is proposed for this purpose. Multicast authentication enables receiver to authenticate the packet source and malicious packets will be denied. There have been many multicast authentication approaches and these approaches could be roughly divided into two categories: symmetric cryptographic primitives and asymmetric cryptographic primitives. Symmetric cryptographic primitive generally uses symmetric key to authenticate data source and MAC (Message Authentication Code) is the well-known approach in this category. In MAC, an identical secret key is maintained at sender and receiver. Sender uses the secret key to generate a MAC for a packet and receiver is able to authenticate the packet source by verifying the MAC of the packet with secret key. Asymmetric cryptographic primitive uses asymmetric key pair to authenticate data source. Asymmetric key pair generally means that there are

two keys, one key is used to generate signature and another key is for verifying the signature. Digital signature is a well-known approach in this category and is believed secure enough. Using RSA to generate digital signature is popular, nevertheless, digital signature generation and verification incur high computation overhead and signing every packet significantly downgrades the system performance. According to this practical concern, a compromised approach which is signature amortization [10][11][15][16][17][18][19][20], was proposed to amortize the overhead of generating one signature over a block of packets. For a block of packets, only one digital signature will be generated. These packets will be considered authentic if the signature of this block can be correctly verified by receiver. Signature amortization makes tradeoff between security and computation overhead. Due to the unreliable transmission and packet loss in multicast protocol, an elaborate signature amortization scheme should be able to work well even though some packets get lost. For this reason, signature amortization schemes with fault-tolerant coding algorithms are proposed. In this kind of scheme, the digital signature for a block of packets is always encoded at sender by fault-tolerant coding algorithms and decoded at receiver. Fault-tolerant coding algorithms like erasure codes [7][8][9][12], or diversity codes [21] partition an information into many segments and the information is able to be correctly reconstructed even though a threshold number of segments get lost. For example, an (n,t) erasure encoder generates a set S of n symbols (s_1, s_2, \dots, s_n) from an information. The erasure decoder can tolerate a loss of up to t packets. Although signature amortization with fault-tolerant coding algorithms reduces computation overhead and tolerates packet loss, it suffers pollution attacks [1]. Pollution attack was first defined in [1] and this attack occurs if attackers inject a great quantity of forged packets into the block of packets. At receiver, these forged packets will disturb the signature decoding procedure of fault-tolerant coding algorithm and then the decoder

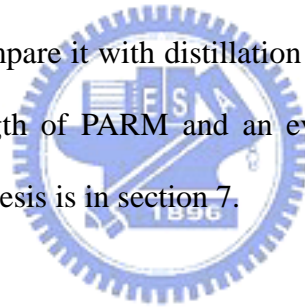
will consequently reconstruct an incorrect signature. The incorrect signature will fail to be verified by receiver with sender's public key and all the packets in the block will be considered invalid. Receiver will drop received packets in the block which includes valid packet transmitted by legal sender. The information inside valid packets will not be understood by receiver; hence the multicast application is unable to serve users fluently.

Distillation codes [1] is so far the only one solution to defend against pollution attack for signature amortization. In distillation codes, sender augments each packet with a witness and receiver is able to partition received packets into many groups according to the augmented witness. Distillation codes guarantees that all valid packets will be partitioned into the same group and correct signature could be decoded from the packets in this group. Therefore, forged packets injected by attackers will be partitioned into other groups and pollution attack is unable to affect the decoding procedure. However, in distillation codes, we can not realize which group contains valid packets in advanced, and every group should execute the decoding procedure to reconstruct the correct signature. Besides, while packets reach receiver side, receiver is unable to determine the received packets valid of invalid instantly and receiver consequently should store all received packets no matter valid or invalid. Distillation codes incurs high computation overhead and the storage space at receiver, and the delay of distillation codes is considerable.

Multicast authentication ensures the security for multicast applications. However, sign every multicast packet with digital signature incurs high overhead and the cumbersome overhead is impractical for many resource limited devices. Signature amortization reduces the computation and communication overhead and a fault-tolerance coding algorithm is always involved in to tolerate packet loss. But a signature amortization scheme with erasure codes suffers pollution attacks which an

attacker can inject forged packets to disturb the decoding procedure of erasure codes. Therefore, to solve this problem, we design a lightweight and pollution attack resistant multicast authentication protocol (PARM). The main advantages of our proposed scheme are fast and lightweight, since many multicast applications are time-sensitive and some end devices may have only limited computation power. Therefore, a high overhead and high delay solution is unsuitable to be widely deployed on multicast applications. In contrast to distillation codes, our proposed scheme requires less computation overhead and less storage space.

In the next section, we briefly discuss related work of signature amortization, and give some overview of a signature amortization scheme, SAIDA, and distillation codes. Our proposed scheme and is given in section 3 and we analyze the overhead of our proposed scheme and compare it with distillation codes in section 4. In section 5, we derive the security strength of PARM and an evaluation is given in section 6. Finally, a conclusion of this thesis is in section 7.



2. Related work

We will introduce current works in signature amortization in section 2.1 and a signature amortization scheme with erasure codes (SAIDA) will be introduced in section 2.2. In section 2.3, we will give a rough description of distillation codes which is proposed to resist pollution attacks in SAIDA,

2.1. Signature amortization

Computation and communication overhead is a significant consideration in many multicast authentication schemes based on digital signature. To reduce the overhead, signature amortization schemes are proposed to amortize the overhead of generating a single signature over many packets. Based on different techniques, signature amortization schemes can be classified into several categories.

Hash chain. Gennaro and Rohatgi in [22] is the first scheme that uses signature amortization over a hash chain. Each packet p_i is augmented with verification information a_i which is recursively defined as the hash value of the concatenation of next packet p_{i+1} and next verification information a_{i+1} . For example, $a_i = h(p_{i+1} || a_{i+1})$ and $a_{i+1} = h(p_{i+2} || a_{i+2})$ where h means a hash function. Since the verification information is used to authenticate next packet recursively, only the first packet with its verification information needs to be signed with digital signature to protect the first verification information against interference. This scheme has constant authentication overhead per packet but does not tolerate packets losses. If one packet gets lost, all continued packets are unable to be authenticated.

Graph-based. Graph-based technique [15][16][18][19] generalizes the idea of

amortizing a signature over a hash chain in such a way as to tolerate packets losses. A single-sink directed acyclic graph (DAG) is defined where each vertex corresponds to a packet. The edges in this graph between two vertices indicate the authentication direction that source vertex is authenticated using the verification information of destination vertex. Instead of augmenting next packet's hash value in current packet, a packet p_i is augmented with the hash value of the packet p_j which points to p_i in single-sink directed acyclic graph. The first packet is also digitally signed. Graph-based schemes just guarantee probabilistic security strength under packet losses occur randomly. In particular, they require that the digitally signed packets need to reach the receiver completely.

Merkle hash trees. Merkle hash tree [6] is a mechanism for computing a single cryptographically secure hash digest over a set of data elements. Merkle hash tree is a binary hash tree and in many signature amortization schemes, Merkle hash tree is build on top of the packets' hash values. The internal nodes are recursively defined as hash values which are produced by hashing the concatenation of its two children.

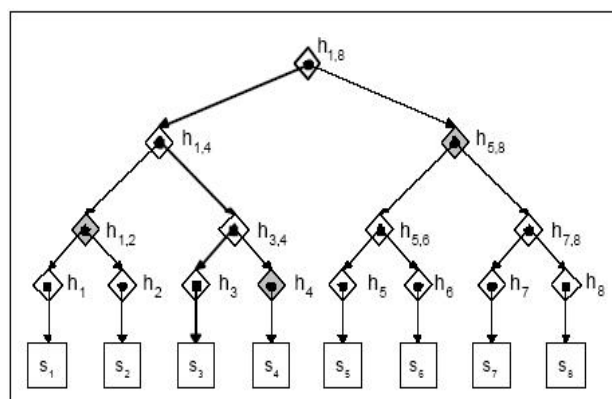


Figure 2-1 Merkle Hash Tree.

Each leaf node is calculated by hashing the corresponding packet S_i , and each internal node is the hash value of the concatenation of its two children. The verification sequence, for example, of a leaf node h_3 is $(h_4, h_{1,2}, h_{5,8})$.

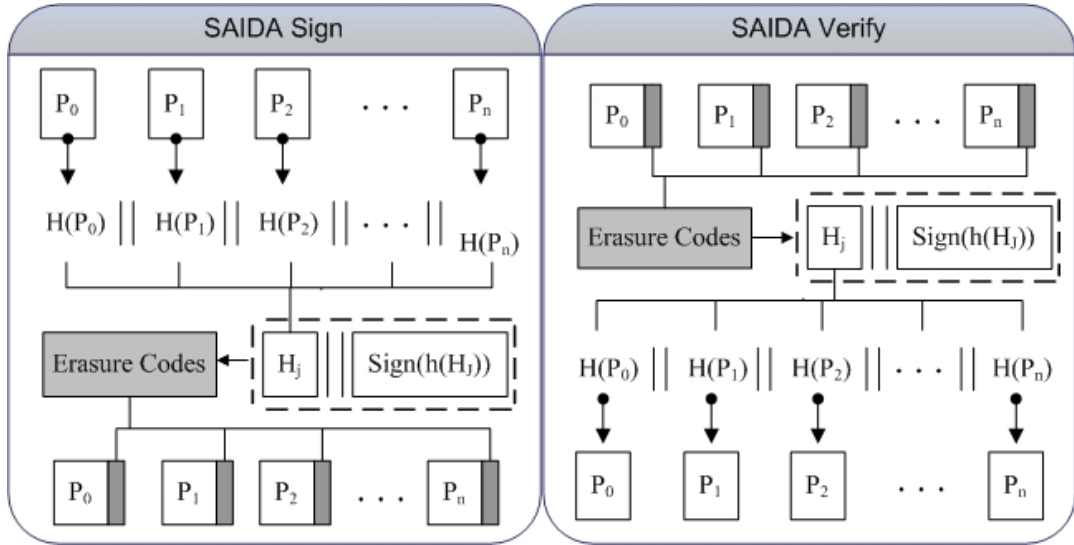
Figure 2-1 shows a Merkle hash tree and each S_i indicates a packet. Each leaf node h_i is calculated by hashing the corresponding S_i , and each internal nodes $h_{i,j}$ means the hash results of the concatenation of h_i and h_j . The verification sequence of a Merkle hash tree for a leaf node indicates the hash values of the sibling nodes on the path from the leaf node to root. With a leaf node and its verification sequence, the root hash value of the tree can be retrieved. For instance, in Figure 2-1, the verification sequence of packet s_3 can be represented as $(h_4, h_{1,2}, h_{5,8})$. In [20], a Merkle hash tree technique is used by Wong and Lam to amortize a digital signature over n packets. Each leaf node of Merkle hash tree means a packet hash value. Each packet is augmented with the verification information which consists of the signed root hash value and corresponding verification sequence of Merkle hash tree. The scheme allows packets to be individually and independently verified. Packet losses also can be tolerated in the scheme, but logarithmic communication overhead per packet exists since the verification sequence size grows logarithmically with the amount of leaf nodes. In contrast, our proposed multicast scheme, which also uses signature amortization, has constant per-packet communication overhead.

Erasure codes. An erasure code [7][8][9][12] is an encoder and decoder that use forward error correction to tolerate data loss. The encoder redundantly encodes information into a set of segments. If the decoder receives sufficiently many segments, it can reconstruct the original information. An (n,t) erasure encoder generates a set S of n segments from the input. The decoder can tolerate a loss of up to t packets. Park et al. [10][11] proposed a signature amortization scheme SAIDA (Signature Amortization using the Information Dispersal Algorithm) which employs the use of erasure codes for multicast authentication to tolerate random packet loss. In [10], a signature is encoded by erasure codes and sender consequently disseminates the encoded signature over a block of packets. Erasure decoder at receiver is able to

reconstruct the signature if receiving sufficient segments. Our proposed scheme is based on SAIDA and we will describe SAIDA more detailed in the next section. However SAIDA is vulnerable to a simple attack that an adversary can disturb the decoding procedure at receiver by injecting forged packets. In [1], they refer to this attack as pollution attack and they proposed a solution named distillation codes to defend against pollution attack where erasure code has been deployed. We will show the techniques and the shortcomings of distillation codes in the following section.

2.2. SAIDA

The full name of SAIDA is signature amortization using the information dispersal algorithm. In SAIDA, the multicast packet stream is partitioned into blocks of n consecutive packets. The hash values of packets in one block are concatenated to form H_j and H_j is protected by generating a signature $Sign(h(H_j))$ for the hash value of H_j . The verification information VI which includes H_j and $Sign(h(H_j))$ is encoded by erasure codes and erasure encoder outputs a set of segments which will be appended to each packet in the block. An erasure code can reconstruct original verification information VI even though some packets get lost. While receiving more than a threshold amount of packets at receiver, receiver is able to decode the VI from these segments of all received packets. After successfully reconstructing VI , receiver should first check the validity of H_j by verifying the $Sign(h(H_j))$. Consequently, the hash values contained in H_j are able to authenticate all received packets in one block.



(a) Signing in SAIDA. All packet values are concatenated to form H_j and a digital signature $Sign(h(H_j))$ is used to protect H_j . Erasure code is able to tolerate packet loss. The output of erasure codes disseminates over all packets in one block.

(b) Verifying in SAIDA. Receiver collects the information segments and use erasure codes to reconstruct H_j and $Sign(h(H_j))$, and receiver verifies $Sign(h(H_j))$ to check the validity of H_j . H_j then is used to validate packets in the block.

Figure 2-2 SAIDA (Signature Amortization using the Information Dispersal Algorithm).

In Figure 2-2(a), P_i denotes a multicast packet and $H(P_0)$ represents the hash value of packet P_0 . H_j means the concatenation of these packets hash values and $Sign(h(H_j))$ is a digital signature for the hash value of H_j . The verification information VI included H_j and $Sign(h(H_j))$ is encoded by erasure codes. Sender will disseminate the output into each packet. Figure 2-2(b) illustrates the SAIDA verifying procedure and the procedure is inverse to SAIDA signing procedure. After reconstructing the concatenation of H_j and $Sign(h(H_j))$, $Sign(h(H_j))$ needs to be verified to check the validity of H_j . Consequently, the hash values included in H_j are used to validate each packet and receiver will accept the packet if its hash values is matched, otherwise, drop it.

Pollution attacks in SAIDA. In normal case, receiver decodes the verification

information from received packet by erasure codes to validate each packet. However when an adversary injects forged packets into communication channel, receiver will be confused and incorrect result will be decoded if there is no packet validation mechanism. Incorrect H_j is unable to be successfully verified by the signature $Sign(h(H_j))$ and all valid received packets will be dropped because these packets can not be authenticated by the hash values included in H_j . If receiver wants to reconstruct the correct information while pollution attack occurs, receiver will spend extreme computation power to try all the possible combinations of received packets to find the correct information.

2.3. Distillation codes

Karlof et al. [1] proposed distillation codes to defend against pollution attacks based on a signature amortization scheme SAIDA. They used Merkle hash tree and one way accumulator to realize distillation codes. First, a Merkle hash tree was constructed using multicast packets hash values. Sender generated witness for a packet and appended the witness to the packet. Witness is the verification sequence of a leaf node in Merkle hash tree and receiver could partition all received packets, which might include attack packets, into many groups according each packet's witness. Distillation codes ensures that there will be a group including all valid packets and the verification information could be successfully reconstructed from packets in this group. The verification information consequently is able to authenticate the packets in one block. Witness plays an important role for distillation codes to resist pollution attacks.

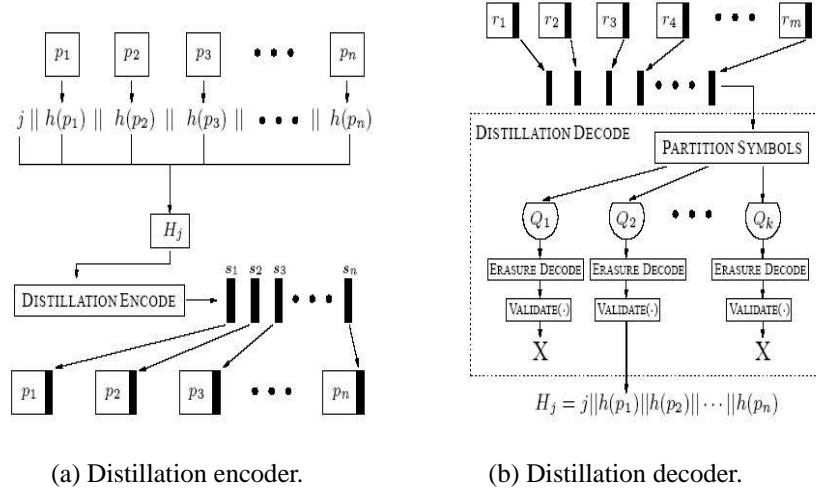


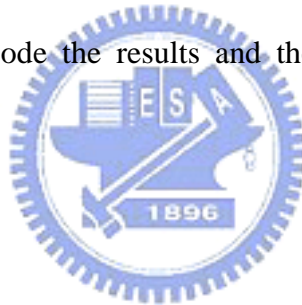
Figure 2-3 Distillation codes.

Figure 2-3 shows the distillation encoder and decoder procedure. In Figure 2-3 (a), packets in one block are first hashed by a hash function and these hash values are concatenated to form H_j . In the distillation encode, H_j is encoded by erasure codes first and the output symbols $S' = (s_1', s_2', \dots, s_n')$ are regarded as leaf nodes to build a Merkle hash tree. The s_l in Figure 2-3 (a) is produced to be the concatenation of s_1' and the verification sequence of s_1' . These packets are augmented with the output of distillation encode symbols (s_1, s_2, \dots, s_n) . Since we can calculate the root hash value of the Merkle hash tree through the verification sequence, in Figure 2-3 (b), receiver is able to partition the received packets according to the calculated root value from the witness of the packet. Since the valid witnesses have the same root hash value and thus the packets with valid witness will be partitioned in the same group. Consequently, receiver can reconstruct correct verification information from the packets in the group and pollution attacks can be defended by distillation codes.

However, distillation codes causes logarithmic communication overhead since witness size grows logarithmically with the number of packets in one block. While suffering pollution attacks, receiver will spend significant computation power since each group should operate erasure decoding and signature verifying to produce the

correct information. In addition, since receiver does not know the correct root hash value of the Merkle hash tree in advanced, the received packets should be buffered no matter invalid or valid until the correct information was reconstructed. This will require large buffer size to store these packets temporarily.

Besides, a weakness occurs in distillation codes. An attacker can construct his own Merkle hash tree and send packets which are augmented with corresponding witness. These packets will be partitioned into same group at receiver since these witnesses are constructed from the same Merkle hash tree. The receiver is not aware of the correct root hash value of Merkle hash tree in advanced, so an attacker is able to inject large number of forged packets into one group to exhaust receiver's computation power. Therefore, in this group, receiver needs to pay lots of computation overhead to decode the results and the performance of receiver will downgrade dramatically.



3. Proposed scheme

To resist pollution attacks, we design a lightweight and pollution attack resistant multicast authentication protocol (PARM) based on SAIDA. The main concept of our proposed scheme is that each packet will be appended an evidence. Evidence means the verification information for a packet to prove the validity of the packet. Evidence of a packet is used for a receiver to judge the validity of a packet. Our proposed scheme is fast and lightweight, since many multicast applications are time-sensitive and some end devices may have only limited computation power. Therefore, a high overhead and high delay solution is unsuitable to be widely deployed on multicast applications.

3.1. Lightweight and pollution attack resistant multicast authentication protocol (PARM)



Our proposed scheme can be roughly divided into four phases: initialization phase, evidence generation phase, evidence validation phase and temporal key renewal phase. We describe the four phases in individual sections.

3.1.1. Initialization Phase

In this phase, we mainly define how to generate temporal key pair, a temporal secret key (TSK) chain and a temporal public key (TPK). One-way hash function is the main technique to generate the temporal key pair. Sender will use TSK chain to generate the evidence of a packet and receiver will validate the evidence of received

packet by TPK.

Sender will generate TSK chain and TPK in advanced before communicating with receivers. First of all, sender generates k random number ($R_0, R_1, R_2 \dots R_{k-1}$) where each random number is n -bits, and these numbers are considered the first TSK of TSK chain, TSK_0 . Let h be a one-way hash function and we will use this one-way hash function to recursively generate other TSKs of TSK chain. Based on existed TSK_0 , TSK_1 is generated by hashing each element in TSK_0 with one-way hash function h and these hash results are collected to form TSK_1 . TSK_1 can be denoted ($h(R_0), h(R_1), h(R_2), \dots, h(R_{k-1})$). According to the same manner, TSK_2 and latter TSKs can be recursively produced by sender, and we assume that totally L TSKs will be generated. That means the TSK chain length is L and TSK chain can be represented as ($TSK_0, TSK_1, TSK_2 \dots TSK_{L-1}$). Temporal public key (TPK) is generated in the same technique that hashing every elements in TSK_{L-1} which is the last TSK of TSK chain with one-way hash function h .

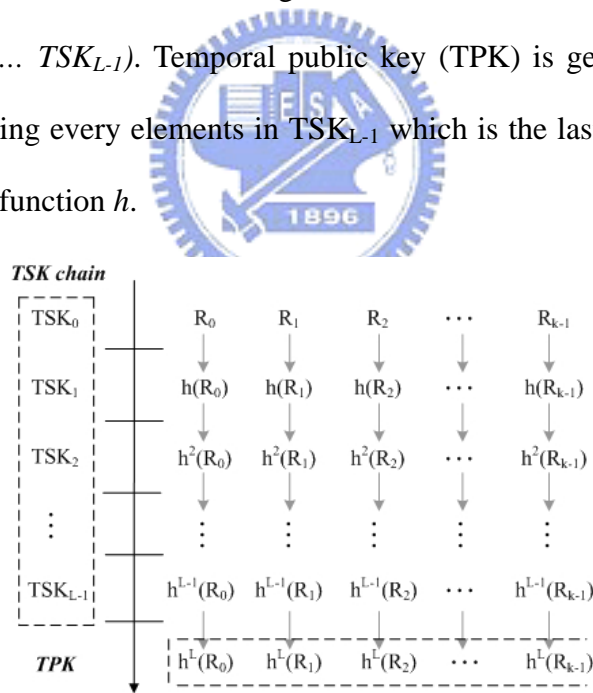


Figure 3-1 Temporal key pair generation.

Figure 3-1 shows the procedure of generating TSK and TPK. For instance, R_0 means the random generated number and the arrows indicate an one-way hash function h . Therefore, $h(R_0)$ indicates the hash result of R_0 , and $h^2(R_0)$ is the hash result of $h(R_0)$. All these TSK elements can be seen as a TSK elements array as shown

in the right part of Figure 3-1. The collection set of the elements in the same row indicates a TSK, for example, $TSK_0 = (R_0, R_1, \dots, R_{k-1})$ and $TSK_1 = (h(R_0), h(R_1), \dots, h(R_{k-1}))$. Each column represents the elements of different TSK with same column index. The elements in the last row are collected to form TPK.

After successfully generating TSK and TPK, sender transfers TPK to receivers. Since receivers will use TPK to determine the validity of received packet, it is very important for receiver to check the correctness of received TPK. If TPK is transferred without any protection, an attacker is able to produce a forged TPK to cheat the receivers. Once receiver accepts the forged TPK which is produced by an attacker, all the valid packets will fail to pass the evidence validation. Consequently, the transmitted TPK should be protected by signing digital signature with sender's private key to prove the data source. After receiving the TPK, receiver will verify the signature to determine if the TPK is transferred by valid sender. If signature verification is correct, TPK will be stored at receiver.

For latter operations, sender should store all these TSK elements and this will be a considerable storage size for sender. However, the role of sender in our multicast environment is always a multicast server which has more computation power and storage size. It is feasible for a server to store the information. The receiver only needs to store TPK for correctly operating consequent phase.

3.1.2. Evidence Generation Phase

When sender wants to send packets to receiver, this phase will be operated to generate the evidence for each packet. Evidence means the verification information for receivers to judge the validity of a packet. Since each packet will be augmented with evidence at sender, the evidence generation phase should be very lightweight and

fast. In evidence generation phase, sender should maintain an appearance times table, and evidence is generated according to this table. The appearance times table counts the appearance times of each column index of TSK elements array. Figure 3-2 represents an example of appearance times table. Each element contained in row *Index* denotes the column index in TSK elements array. Each number in row *Appearance times* shows the used times of the corresponding index in evidence generation phase. All elements of each TSK are considered a TSK elements array, and the row x of this array indicates TSK_x and column index y indicates the elements which is the y -th in each TSK.

Index	0	1	2	3	4	...	k-2	k-1
Appearance times	1	5	3	8	0	...	0	2

Figure 3-2 Appearance times table.

Assume the packet size is m -bits, and to generate evidence for this packet, we first hash the packet with one-way hash function h . The hash value is divided into p segments, where each segment size is b -bits, and each segment is interpreted as an integer between 0 and 2^b-1 . We denote these p segments $S = (i_0, i_1, \dots, i_{p-1})$. Each segment in the set S stands for the column index of TSK elements array. For each index i , the appearance times of index i is used to determine which TSK will be selected and index i means the i -th element should be chosen in selected TSK. If the appearance times of index i is a_i , then $TSK_{(L-1)-a_i}$ will be selected,. Therefore, if index i never appears before, the last TSK, TSK_{L-1} , of TSK chain will be selected. For example, if the appearance times of i_0 is 4, then sender will choose the i_0 -th element in TSK_{L-5} . Each index in $(i_0, i_1, \dots, i_{p-1})$ will be operated using above manner and consequently p elements of TSK will be produced for each index. These p elements

are composed to form the evidence of the input packet and the evidence is then appended to the packet. Sender transfers the packet with its evidence to receivers and receivers are able to validate the evidence to determine the validity of this packet.

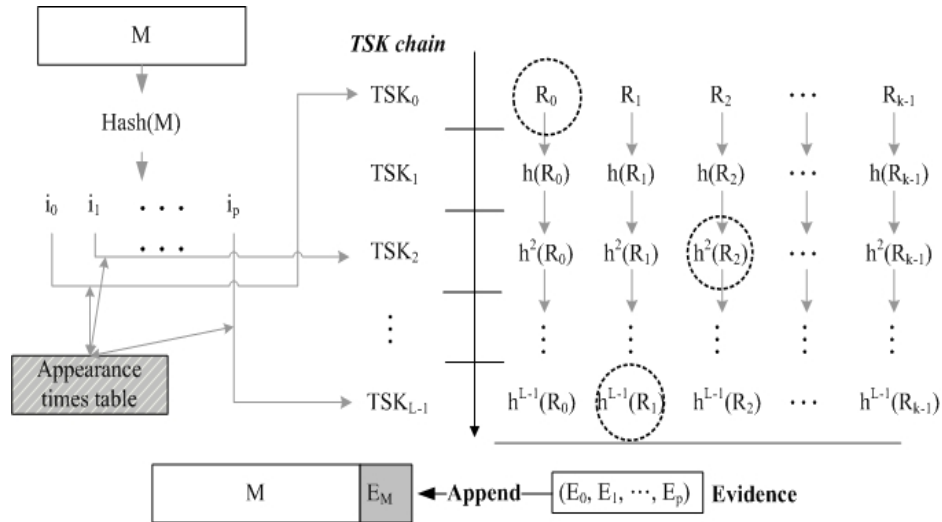


Figure 3-3 Evidence generation phase.

Figure 3-3 shows a illustration for evidence generation phase. In this figure, i means the index number of TSK elements array and E_M indicates the evidence for a packet M . For example, i_0 is zero and the appearance times of i_0 is $L-1$, and thus R_0 is selected. Each index i and its corresponding appearance times are used to select the relative elements and these elements are concatenated to form the evidence E_M .

Note that the evidence generation in our proposed scheme is very lightweight and fast, since it needs no complicated operation to generate evidence but few one-way hash operations. Lightweight and fast evidence generation is a significant feature to raise the performance of multicast applications.

3.1.3. Evidence Validation Phase

While receiving a packet, receiver can validate the evidence attached to the

packet to check the packet valid or invalid. If an attacker wants to forge a packet to cheat receiver, he must have the ability to generate the valid evidence for the forged packet. It is very hard to generate the evidence for forged packet without the knowledge of TSK, and we will show the complexity for an attacker to successfully cheat the receiver in section 5. At receiver, temporal public key (TPK) is used to validate the evidence. As the sender, receiver also needs to maintain an appearance times table for each column index of TSK elements array based on received packets.

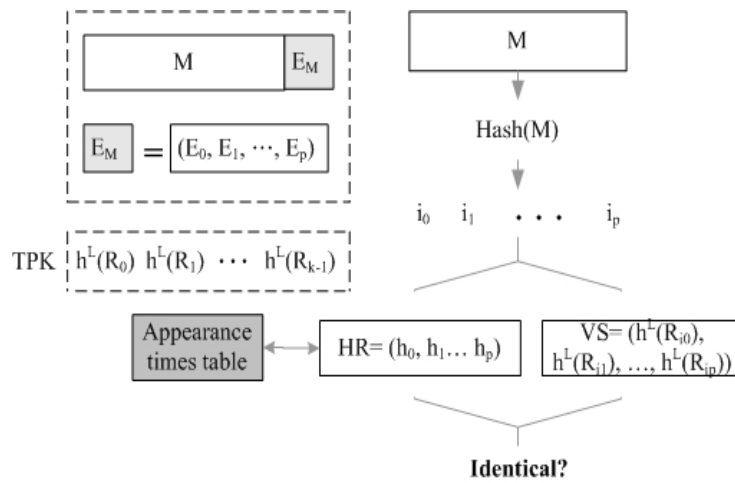
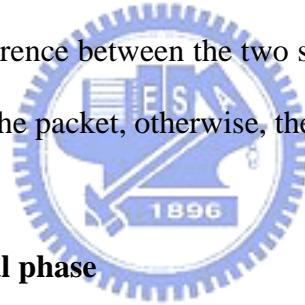


Figure 3-4 Evidence validation phase.

We illustrate the evidence validation phase in Figure 3-4 Evidence validation phase.. As shown in this figure, the evidence validation procedure at receiver is a little similar to evidence generation phase at sender. Assume the receiver has received a packet with its evidence, and to validate the evidence for this packet, receiver first hashes the packet M with one-way hash function h which is identical to the one-way hash function used in evidence generation phase at sender. We denote the received evidence $E = (e_0, e_1, \dots, e_p)$. The hash value $h(M)$ is divided into p segments where each segment size is b -bits and each segment is interpreted as an integer between 0 and $2^b - 1$. We denote these p segments $(i_0, i_1, \dots, i_{p-1})$. Each segment in this set stands for the

column index of TSK elements array. For each index i , the appearance times of index i is used to individually determine the number of times to hash the elements in evidence. For instance, if the appearance times of index i_j is a_i , the corresponding element e_j in evidence should be hashed $ai+1$ times. According to the same manner, if index i_j never appears before, the element e_j in evidence only needs to be hash one times. Each element in evidence E will be hashed for corresponding times according to above-mentioned manner. The hash results of the elements can be denoted $HR = (h_0, h_1, \dots, h_p)$. Receiver consequently choose the verification subset $VS = (h^L(R_{i0}), h^L(R_{i1}), \dots, h^L(R_{ip}))$ from TPK where $h^L(R_{ij})$ means the ij -th element in TPK. Receiver compare the two sets, HR and VS , and if all elements are identical between these two set, the evidence will be considered valid. On the contrary, the evidence will be considered invalid if any difference between the two sets. If the evidence validation is correct, receiver will accepts the packet, otherwise, the packet will be dropped.



3.1.4. Temporal key renewal phase

With previous three proposed phases, sender can generate the evidence using TSK and receiver is able to judge the validity of packets by verifying the evidence with TPK. The evidence of packet prevents receiver from accepting forged packet. Therefore, pollution attacks will not occurs in our proposed scheme. However, in evidence generation phase, each evidence includes some elements of TSK, and an attacker can sniff the network to obtain these elements contained in evidence. When an attacker obtains enough portions of TSK elements, the probability of forging valid evidence will rise dramatically, and it means the security strength decreases as the portion of TSK an attacker has obtained increases. Thus periodically renew the used TSK elements is necessary to ensure the secure communication between sender and

receivers. We first define a threshold value T in our key renewal phase. U_{TSK_0} denotes the numbers of used elements in TSK_0 since the last time temporal key renewal phase has been operated till now. TSK_0 stands for the first TSK of TSK chain. The occasion to execute temporal key renewal phase is when the numbers of used elements in TSK_0 , U_{TSK_0} , has exceeded the threshold T . In this phase, assume the indexes of these used elements in TSK_0 are denoted $(j_0, j_1, j_2 \dots j_{t-1})$. Sender consequently generates t new random numbers for these indexes in TSK_0 . Based on these random numbers, we use one-way hash function h to generate other elements in TSK_j , and new partial TPK also be generated. The above-mentioned operation is the same as the temporal key generation procedure in initialization phase. Since we only partially generate new TSK, we name this new information partial TSK. We update original TSK elements with partial TSK elements, whose column index is in $(j_0, j_1, j_2 \dots j_{t-1})$. The original TPK in receiver also needs to be updated to cooperate with the renewed TSK. The new partial TPK will be concatenated with H_j and $Sign(h(H_j))$ in SAIDA to form a new verification information VI' . The new verification information is then encoded by erasure codes as usual in SAIDA and the output segments are appended to each packet. Figure 3-5 gives an illustration of above-mentioned procedure. As receiver successfully reconstructs the VI' original TPK stored in receiver can be updated with the new partial TPK contained in VI' .

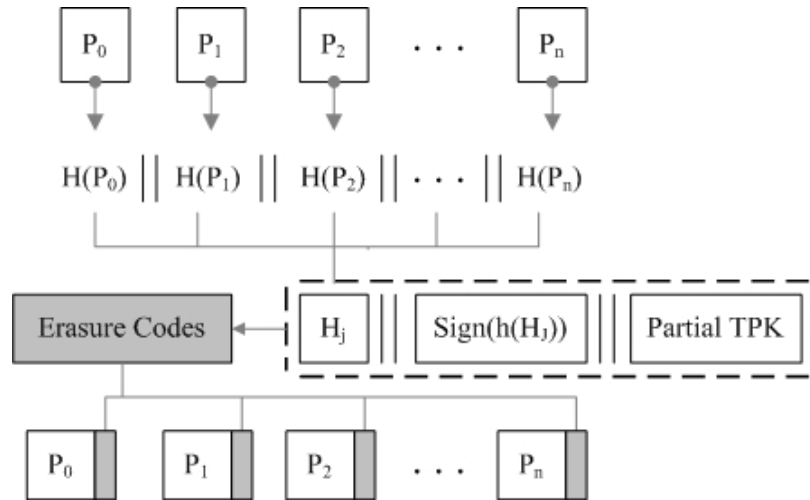
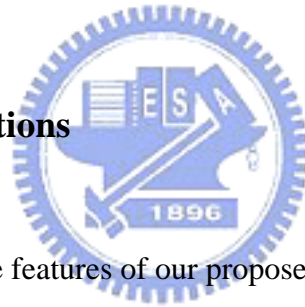


Figure 3-5 Temporal key renewal phase.

After successfully renewing TSK and TPK, sender and receiver are able to use new TSK and TPK to generate and validate evidence.

3.2. Practical considerations



In this section, we list the features of our proposed scheme in the following:

Efficient evidence generation and validation. Our proposed PARM uses one-way hash function to generate and validate evidence instead of conventional complicated cryptography algorithm. PARM is also suitable for those devices with restricted computation power due to the advantage of lightweight operations.

Instant validation. Receiver is able to validate each packet based on appended evidence once a packet is arriving. Thus, no redundant storage space is required to buffer invalid packets. Instant validation also prevents attacker overloads receiver by sending large amounts of packets to overwhelm the receiver's storage space.

Packet loss tolerant and individual validation. Since in many multicast

applications, lost packets are not retransmitted, our proposed scheme PARM can tolerate up to a threshold number of packet loss. Moreover, packet loss will not affect the validation of other packets that each packet is able to be validated individually and independently. The feature is very important for a multicast authentication scheme since packet loss occurs frequently in Internet.

Constant verification information (Evidence). In order to validate individual packets instantly, every packet will be augmented with its own evidence. If each evidence size is too large or the evidence grows with the scale expanded, the overhead of evidence will significantly affect the performance. In our proposed scheme, the evidence size remains constant per-packet communication overhead.

3.3. Attacks resistance



There are many kinds of attacks in Internet and a considerate multicast authentication scheme should be able to defend against kinds of attacks. In this section, we show that how PARM could resist common attacks. Here we assume an adversary is infeasible to successfully forge evidence, and we will show the degree of difficulty for this assumption in section 5.

Injection. Attackers always inject random or pre-designed packets to cheat the receiver and may induce the receiver to make illegal behavior. This attack will not gain its purpose since each valid packet will bring corresponding evidence and receiver will accept the packets only if the evidence validation is correct.

Modification. Another often seen attack is modification that attacker captures

transmission packets and retransfers it after modifying the content. The evidence of a packet is related to the packet content, thus the modification will be aware for the receiver by validating the evidence.

Signature flooding attacks. Since almost all authentication mechanism uses additional verification information for a receiver to validate the received packets. However, an adversary may realize signature flooding attacks by sending large amount of packets with invalid verification information. If the validation operation is high overhead, receiver consequently exhaust its resources to validate those invalid verification information. Nevertheless, our proposed PARM is resistant to this attack due to the lightweight validation procedure of our scheme.

Pollution attacks. To pollute the erasure decoding procedure is infeasible because each valid packet should be appended with evidence which will be validated by receiver. Thus, only valid packets will be accepted by receiver and pollution attack is unable to be realized under our proposed scheme.

4. Comparison

A distillation code proposed by Karlof et al was so far the one which has solved the pollution attacks in SAIDA. However, distillation codes needs more communication and computation overhead and we make a comparison between distillation codes and our proposed scheme (PARM).

Before the comparison, we first define some parameters to complete the comparison.

C_E : computation overhead of operating erasure codes per time.

C_H : computation overhead of operating hash functions per time.

S_G : computation overhead of generating one digital signature in SAIDA.

S_V : computation overhead of verifying one digital signature in SAIDA.

N_K : number of packets in one SAIDA block.

N_P : verification information size of our proposed scheme.

N_A : total amount of attack packets in one block.

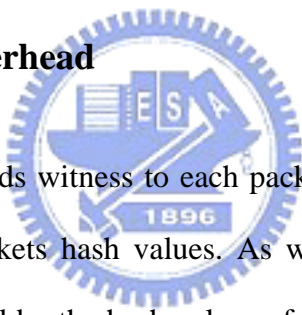
4.1. Storage overhead

In the initial stage, our proposed scheme requires to store some information in sender and receiver, and Distillation codes requires no additional storage size. In our proposed scheme, assume the TSK chain length is L and each TSK contains k elements, then the size of TSK elements array is $L*k$ and TPK size is k . Therefore, sender has to store $L*k$ elements and receiver has to store k elements. If the TSK chain length L is very large, then the storage overhead at sender will be impressive. However, the environment of this paper is multicast and the sender means a multicast server which generally has larger storage size and higher computation power. Thus the

storage size consumption will not cause significant overhead to sender.

While pollution attack occurs in multicast communication, our proposed scheme is able to save considerable storage space in contrast to Distillation codes. Since our proposed scheme check the validity of received packets instantly, only valid packets will be buffered. However Distillation codes can not judge the correctness of received packets and all packets, no matter valid or invalid, should be buffered to be used in the followed procedure. Due to the characteristic of limited resource at receiver, buffering amount of invalid packets impressively affect the receiver's performance. Therefore, our PARM is more efficient than Distillation codes at receiver side while pollution attack occurs.

4.2. Communication overhead



A distillation code appends witness to each packet and the witness is generated by Merkle hash tree and packets hash values. As we described in related work, a Merkle hash tree is established by the hash values of the block of packets in SAIDA, and the witness of a packet means the verification sequence of the packet in this tree. However verification sequence of Merkle hash tree increases in logarithmic to the number of leaf nodes. Therefore, if the number of packets in one block is scaled, the witness size is consequently increased in logarithmic.

In contrast to distillation codes, our proposed scheme uses constant size evidence no matter number of packets in one block. Thus, our proposed scheme has lightweight communication overhead than distillation codes.

4.3. Computation overhead

In this section, we would compare the computation overhead with Distillation Codes.

We first show the computation overhead at receiver and sender to send or receiver one block packets when there is no pollution attack. When Distillation Codes is deployed, the computation overhead of sending one block packets at sender could be denoted

$$(2N_K - 1) * C_H + S_G.$$

The computation overhead of receiving one block packets at receiver could be denoted

$$N_K * (\log_2 N_K + 1) * C_H + C_E + S_V.$$

In contrast to Distillation codes, our proposed scheme costs

$N_K * C_H + S_G$ computation overhead at sender. And at receiver, the computation overhead to validate received packets in one block could be denoted

$$N_K * N_P * C_H + C_E + S_V.$$

The computation costs constant overhead in our proposed scheme and costs logarithmic overhead in Distillation codes. Because computation overhead would not be affected at sender when happening attacks, we then only analyze the computation overhead at receiver when pollution attacks occurred. Some attack parameters should be defined in prior.

Distillation codes costs

$$(N_K + N_A) * (\log_2 N_K + 1) * C_H + N_G * C_E + N_G * S_V$$

computation overhead at receiver where N_G denotes the number of partitions

Distillation codes would partition. Compared to our proposed scheme, the overhead might be derived as

$$(N_K + N_A) * N_P * C_H + C_E + S_V.$$

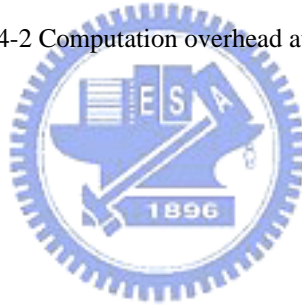
Erasure codes and signature verification cost high computation power and Distillation codes requires several computation times of these two computation-consumed operations. Thus, our proposed scheme is much lightweight than Distillation codes.

	Distillation codes	Proposed scheme
No pollution attacks	$(2N_K - 1) * C_H + S_G$	$N_K * C_H + S_G$
Under pollution attacks		

Figure 4-1 Computation overhead at sender.

	Distillation codes	Proposed scheme
No pollution attacks	$N_K * (\log_2 N_K + 1) * C_H + C_B + S_V$	$N_K * N_P * C_H + C_B + S_V$
Under pollution attacks	$(N_K + N_A) * (\log_2 N_K + 1) * C_H + N_G * C_B + N_G * S_V$	$(N_K + N_A) * N_P * C_H + C_B + S_V$

Figure 4-2 Computation overhead at receiver.



5. Security analysis

We now analyze the degree of computational difficulty to compute the valid evidence without gaining any elements in TSK. Assume each TSK elements is b -bits, the amount of hash values that an adversary has to guess is $(2^{b-1})^p$ on average. The complexity of finding a valid evidence of a packet can be denoted as

$$\text{complexity} = O(2^{bp})$$

where b denotes number of bytes for each symbol and p denotes number of symbols in each evidence.

If an adversary has obtained n TSK elements, we then derive the probability that the valid evidence can be produced. Since each element in TSK will not be reused except those elements in the first TSK chain, TSK_0 , we can assume that only the elements in TSK_0 an adversary has known will affect the security PARM. If these elements an adversary has obtained are normally distributed among each TSK, the amount of these symbols in TSK_0 can be roughly denoted n/L . Assume the TSK chain length is L and each TSK has k elements. The evidence of each packet contains p TSK elements. Under no temporal key renewal, we derive that a tight upper bound of the probability P_f for an adversary to forge a valid evidence using obtained n TSK elements.

$$P_f = \left(\frac{\frac{n}{L}}{k}\right)^p = \left(\frac{n}{Lk}\right)^p \quad (\text{equation 5.1})$$

And we define the security strength as

$$S = \frac{1}{P_f} = \left(\frac{Lk}{n}\right)^p \quad (\text{equation 5.2})$$

The equation shows that the security strength S increased in inverse proportion to

amounts of used TSK elements n and evidence size p . The S increased in direct proportion to amount of TSK elements k and TSK chain length L . The security strength S gets down if probability P_f gets high. In other words, it is securer if an adversary obtained less TSK elements or extending the evidence size.

Once the temporal key renewal phase is deployed, the TSK elements attacker has obtained would be useless to forge the evidence. T means the threshold which is defined in temporal key renewal phase. The upper bound probability of forging evidence can be derived as

$$P_f = \left(\frac{\frac{n}{L}}{k} \right)^p \quad \frac{n}{L} \leq T \quad (\text{equation 5.3})$$

The security strength can be represented as

$$S = \frac{1}{P_f} = \left(\frac{Lk}{\frac{n}{L}} \right)^p \quad \frac{n}{L} \leq T \quad (\text{equation 5.4})$$

We denote n/L to be number of elements in TSK_0 which has been used. If n/L is more than T , n/L would be set to zero since temporal key would be renewed and there would be no more used elements in TSK_0 .

6. Evaluation

In this section, we will make some evaluations to show the security strength of our proposed scheme under different conditions. According to previous derived equations in section 5, we will vary the parameter value to perform the change of security strength.

First, we discuss the situation while no key renewal is adopted. Equation 5.2 in section 5 shows the security strength. We vary the evidence size and illustrate the change of security strength while attacker obtained different number of TSK elements in the following two figures. In Figure 6-1 and Figure 6-2, we assume the parameters in equation 5.2 as follows: each TSK contains $k=512$ symbols and TSK chain length is $L=10$. Each evidence contains p TSK elements. These two figures show the relationship between the security strength and number of TSK elements attacker has obtained.

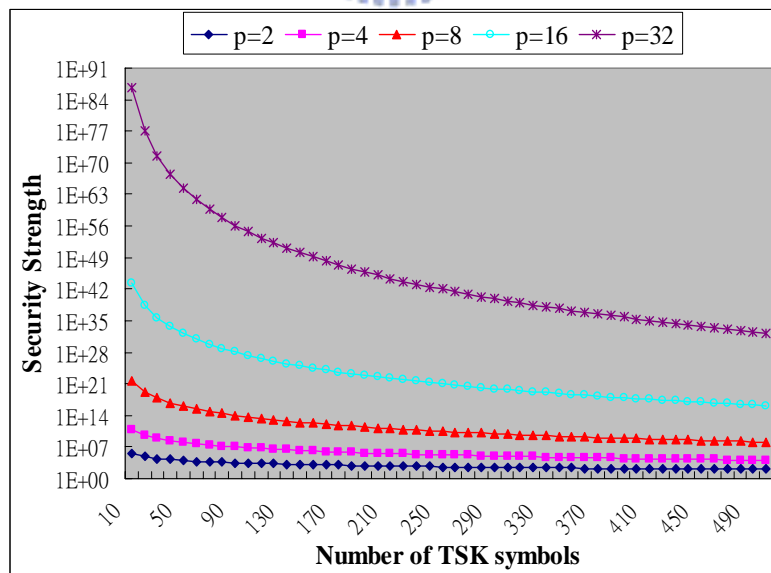


Figure 6-1 The security strength of different evidence size (1).

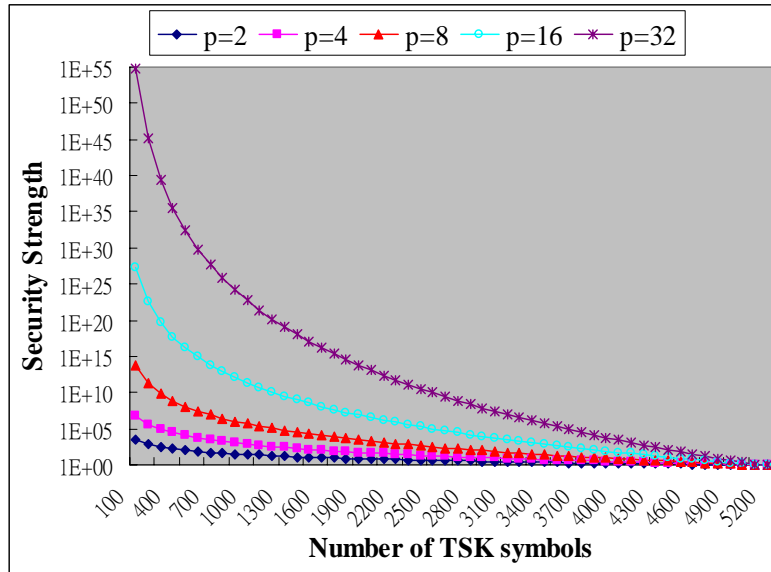


Figure 6-2 The security strength of different evidence size (2).

The y-axis represents the security strength and the x-axis represents the number of TSK elements attacker has obtained. The y-axis is denoted as a logarithmic axis for conveniently displaying the values. In Figure 6-1, the total TSK symbols are 5120 and we show the probability when an adversary obtained 10 to 500 TSK symbols. We could see that the security strength gets higher when evidence size p is bigger. Since temporal key renewal phase is not deployed, the probability would exponentially increase when number of TSK elements attacker obtained increased. Figure 6-2 illustrates the security strength of our proposed scheme when more TSK elements have known by attacker. The number of TSK elements is from 100 to 5120. We could find out the phenomenon that the security would significantly downgrade when there are much TSK elements disclosed.

TSK chain length L is also an important factor to security strength. Figure 6-3 represents the relations while different chain lengths are used. In Figure 6-3, we assume the evidence size p is 16 and each TSK contains $k= 512$ elements. This figure shows the relationship between the security strength or PARM and different TSK chain lengths.

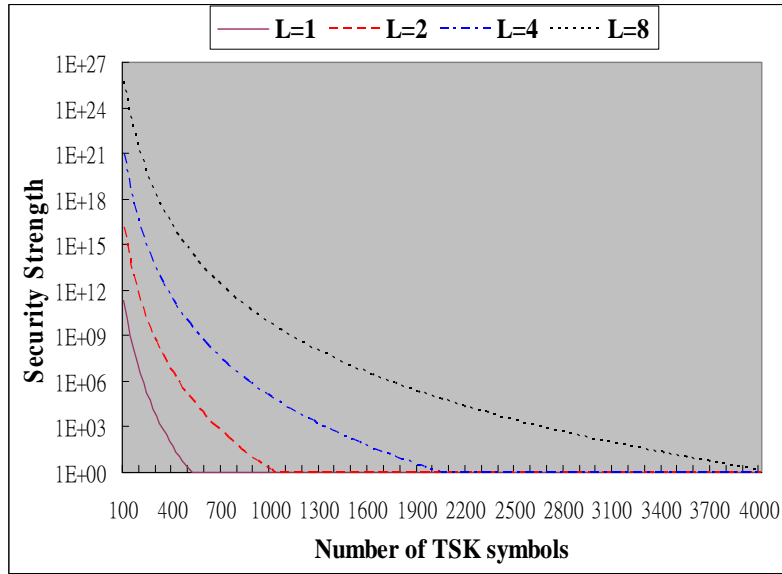


Figure 6-3 The security strength of different TSK chain length.

Based on the equation 5.2, we could obviously find out that TSK chain length has exponential effect to the security strength of proposed scheme, and longer TSK chain length refers to higher security. These figures shown above did not deploy the key renewal mechanism and the security strength will drop dramatically with the number of disclosed TSK elements increased.

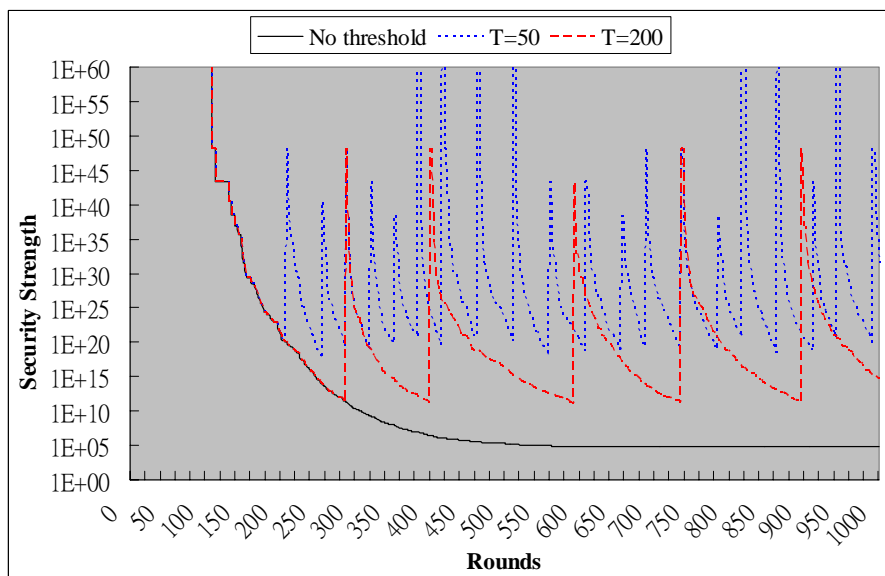
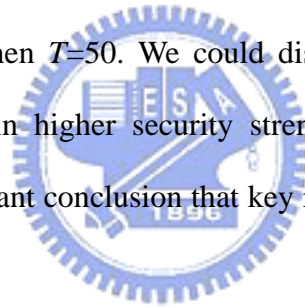


Figure 6-4 The security strength with key renewal.

To enhance the security, we proposed the key renewal mechanism and equation

5.4 represents the security strength when deploying this mechanism. Temporal key renewal phase occurs while the number of used TSK elements in TSK_0 reaches a threshold T and those used TSK elements and TPK would be partially renewed. We then illustrate the equation to show the enhancement of security after deploying key renewal mechanism in our proposed scheme. In Figure 6-4, we show an evaluation of our proposed scheme with different key renewal threshold T . The x-axis represents the number of transmission rounds between sender and receivers. The first curve didn't deploy key renewal mechanism and thus the security strength drops continuously. The other curves deploy temporal key renewal mechanism and obviously they are able to sustain above certain security strength. For example, when threshold T is set to 200, the security strength can be always higher than $E+11$, and the security strength can be always higher than $E+19$ when $T=50$. We could discover from the figure that the small threshold would sustain higher security strength than large threshold. This figure also gives us an important conclusion that key renewal is essential to guarantee the security.



7. Conclusion

Pollution attack is a significant problem in multicast authentication, but previous researches solved the problem without efficiency. This paper has proposed a new approach to resisting pollution attack. This approach not only demanded lightweight computation overhead for sender and receiver but also allowed receiver to instantly validate packets without buffering invalid packets. The idea of partial key renewal in proposed scheme guaranteed a lower bound security regardless of amount of disclosed TSK elements. In addition to SAIDA, the proposed approach could be also used in other signature amortization schemes which had involved fault-tolerant algorithm in to defend against pollution attack.

We derived some overhead analyses and evaluations, and the results shows the proposed scheme is relatively lightweight than previous solutions. The evaluation of threshold of partial key renewal can help sender define the value suitable for local network. Our future work is to reduced the communication overhead of PARM to achieve same security strength and reduce the storage overhead for the considerable requirement of storage size to store TSK elements.

References

- [1] Chris Karlof, Naveen Sastry, Yaping Li, Adrian Perrig, and J.D. Tygar. Distillation Codes and Applications to DoS Resistant Multicast Authentication. In Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04), February 2004.
- [2] J. M. Park, E. Chong, and H. J. Siegel. Efficient multicast packet authentication using erasure codes. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):258–285, May 2003.
- [3] J. M. Park, E. K. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 227–240, May 2002.
- [4] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In Proceedings of the Eighth ACM Conference on Computer and Communications Security (CCS-8), pages 28–37, Philadelphia PA, USA, Nov. 2001.
- [5] L. Reyzin and N. Reyzin. Better than BiBa: Short onetime signatures with fast signing and verifying. In Seventh Australasian Conference on Information Security and Privacy (ACISP 2002), July 2002.
- [6] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134, Apr. 1980.
- [7] M. Luby. LT codes. In 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '02), 2002.
- [8] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, February 2001.
- [9] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault

- tolerance. *Journal of ACM*, 36(2):335–348, 1989.
- [10] J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In *IEEE Symposium on Security and Privacy*, pages 227–240, 2002.
- [11] J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using erasure codes. *ACM Transactions on Information and System Security*, pages 6(2):258–285, May 2003.
- [12] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.
- [13] H. Krawczyk. Distributed fingerprints and secure information dispersal. In *13th ACM Symposium on Principles of Distributed Computing*, pages 207–218. ACM, 1993.
- [14] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134, Apr. 1980.
- [15] P. Golle and N. Modadugu. Authenticating streamed data in the presence of random packet loss. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS 2001)*, pages 13–22. Internet Society, Feb. 2001.
- [16] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 232–246, May 2001.
- [17] A. Pannetrat and R. Molva. Efficient multicast packet authentication. In *Proceedings of the Symposium on Network and Distributed System Security Symposium (NDSS 2003)*. Internet Society, Feb. 2003.
- [18] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signature of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 56–73, May 2000.

- [19] D. Song, D. Zuckerman, and J. D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 258–270, May 2002.
- [20] C. Wong and S. Lam. Digital signatures for flows and multicasts. In *Proceedings on the 6th International Conference on Network Protocols (ICNP '98)*, pages 198–209. IEEE, October 1998.
- [21] E. Ayanoglu, I. Chih-Lin, R.D. Gitlin, J.E. Mazo. Diversity Coding for Transparent Self-Healing and Fault-Tolerant Communication Networks. *IEEE Transactions on Communications*, 41(11), 1993.
- [22] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Advances in Cryptology*, volume 1294 of *Lecture Notes in Computer Science*, pages 180--197. Springer, 1997.

