行政院國家科學委員會專題研究計畫　期中進度報告

安全與可信賴之微型監測器系統網路設計與實作(2/3)

計畫類別：個別型計畫
計畫編號：NSC93-2213-E-009-054-
執行期間：93 年 08 月 01 日至 94 年 10 月 25 日
執行單位：國立交通大學資訊工程學系(所)

計畫主持人：謝續平

報告類型：完整報告
報告附件：出席國際會議研究心得報告及發表論文
國際合作計畫研究心得報告
處理方式：本計畫可公開查詢

中　華　民　國 95 年 1 月 23 日

行政院國家科學委員會專題研究計畫期末報告

具安全通訊與系統防護之無線微型感測網路研發與實作

計畫編號： 93-2213-E-009-120-
執行期間： 93 年 8 月 1 日 到 94 年 7 月 31 日
主持人：謝續平
執行單位：國立交通大學資訊工程學系

Abstract

Authentication is an important security measure for multicast applications, providing receivers with confidence that the packets they receive are valid. Simply signing every multicast packet with a digital signature incurs high overhead; therefore, a scheme such as signature amortization helps reduce this overhead. To tolerate packet loss, erasure codes are employed to enhance signature amortization. However, the use of erasure codes introduces pollution attack, an attack in which the adversary injects packets to disrupt the erasure decoding procedure and consequently denies the authentication service to the receiver. Unfortunately, current solutions to pollution attack are computationally intensive and inefficient. To cope with this problem, we propose a new lightweight, pollution-attack resistant multicast authentication scheme (PARM), which generates evidence that receivers can validate on a fast, per-packet basis. This approach effectively resists pollution attacks and has better performance than previously proposed solutions.

1. Introduction

A 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, e.g. video transmissions, live broadcasts, stock quotes, or news feeds. These applications may have many receivers or distribute time-sensitive data. To ensure secure communications between a sender and its receivers, it is important to implement security measures in a multicast environment.

An attacker may impersonate a sender to transmit malicious packets to receivers, causing disruptions in the communications. Multicast authentication is used to defend against forged packets injected by the attackers by enabling a receiver to authenticate the packet source and discard malicious packets. There have been many multicast authentication approaches, which can be roughly divided into two categories: symmetric cryptographic primitives and asymmetric cryptographic primitives. Symmetric cryptographic primitives, such as MAC (Message Authentication Code), generally use a symmetric key to authenticate a data source. In MAC, an identical secret key is maintained by the sender and receiver. The sender uses the secret key to generate a MAC for a

packet, and the receiver is able to authenticate the packet source by verifying the MAC of the packet with the secret key. Asymmetric cryptographic primitives, such as digital signatures, use an asymmetric key pair to authenticate a data source. In general, an asymmetric key pair consists of two keys; one key is used to generate the signature, while the other key is for verifying the signature. Using digital signatures, like RSA, for authentication are popular and believed to be secure; nevertheless, digital signature generation and verification incur significant computation overhead.

Signature amortization [10][11][15][16][17][18][19][20] addresses this concern by generating a single digital signature for a block of packets. After verifying the signature, a receiver can consider this block of packets authentic. Signature amortization makes a tradeoff between security and computation overhead. An elaborate signature amortization scheme should still work well despite packet loss in a multicast protocol. For this reason, signature amortization schemes utilize fault-tolerant coding algorithms to encode and decode packets. Fault-tolerant coding algorithms, like erasure codes [7][8][9][12] or diversity codes [21], partition information into many segments and can correctly reconstruct the original information even though up to a threshold number of segments are lost.

Although signature amortization with a fault-tolerant coding algorithm reduces computation overhead and tolerates packet loss, it suffers from pollution attacks [1]. Pollution attack occurs when an adversary injects a large quantity of forged packets into a block of valid packets. The receiver fails to decode a correct signature using the fault-tolerant coding algorithm, forcing the receiver to drop the entire block of packets, including valid packets.

Distillation codes [1] have been proposed for signature amortization to defend against pollution attack. In distillation codes, the sender augments each packet with a witness. Upon receipt, the receiver separates packets into groups by witness. Distillation codes guarantee that all valid packets are partitioned into groups that do not contain forged packets, allowing the receiver to decode the correct signature from the packets in this group. Unfortunately, the receiver cannot realize in advance which group contains valid packets; therefore, it must attempt to decode a valid signature from each group. Furthermore, the receiver cannot immediately distinguish between valid and invalid packets, and thus, must first buffer all received packets. Distillation codes incur high computation overhead, storage space, and verification delay.

To summarize, authentication in multicast applications is an important security measure which cannot be neglected. However, signing every multicast packet with a digital signature incurs high overhead, which may be impractical for many resource limited devices. Signature amortization can reduce the computation and communication overhead, and a fault-tolerance coding algorithm can help tolerate packet loss. Despite these countermeasures, a signature amortization scheme still suffers pollution attack. To solve this problem, we design a lightweight and pollution attack resistant multicast authentication protocol (PARM). Our proposed scheme is fast and lightweight, which is ideal for multicast applications with time-sensitive requirements or devices with limited computational power. In contrast to distillation codes, our proposed scheme requires less computation overhead and storage space.

In the next section, we briefly discuss work related to signature

amortization, an overview of the SAIDA signature amortization scheme, and distillation codes. We describe our proposed scheme in section 3 and provide an analysis of PARM against distillation codes in section 4. After deriving the security strength of PARM in section 5, we evaluate its security in section 6. Finally, we conclude our findings in section 7.

2. Related work

We introduce current works in signature amortization in section 2.1 and a signature amortization scheme with erasure codes (SAIDA) in section 2.2. We then give a description of distillation codes in section 2.3.

2.1. Signature amortization

Computation and communication overhead is a significant consideration in many multicast authentication schemes based on digital signature. To reduce this overhead, signature amortization schemes generate a single signature over many packets. Based on different techniques, signature amortization schemes can be classified into several categories: hash chains, graphs, Merkle hash trees, and erasure codes.

Hash chains. Gennaro and Rohatgi in [22] devise a signature amortization over hash chains. Each packet p_i is augmented with verification information a_i, which is recursively defined as the hash value of the concatenation of the next packet p_{i+1} and the 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 denotes a hash function. Since the verification information is used to authenticate the next packet recursively, only the first packet with its verification information needs to be signed with a digital signature to protect against tampering. This scheme has constant authentication overhead per packet but does not tolerate packet losses since the loss of one packet prevents authentication of the remaining packets.

Graphs. A graph-based technique [15][16][18][19] generalizes the idea of amortizing a signature over a hash chain to tolerate packet loss. A single-sink directed acyclic graph (DAG) is defined such that each vertex corresponds to a packet. The edges in this graph indicate the authentication direction; thus, the source vertex is authenticated using the verification information of the destination vertex. Instead of augmenting the current packet with the next packet's hash value, a packet p_i is augmented with the hash value of the packet p_j, which points to p_i in a single-sink DAG. As with the hash chain approach, the first packet is also digitally signed. Graph-based schemes simply guarantee probabilistic security strength under random packet loss. In particular, they require that the digitally signed packets completely reach the receiver.

Merkle hash trees. A Merkle hash tree [6] is a mechanism for computing a single cryptographically secure hash digest over a set of data elements. It is a binary hash tree whose internal nodes are recursively defined as the hash value of the concatenation of its two children. Many signature amortization schemes build a Merkle hash tree on top of the packets' hash values.

Figure 2-1 shows a Merkle hash tree with each S_i representing a packet. Each leaf node h_i is calculated by hashing the corresponding S_i, while each internal node $h_{i,j}$ is computed by hashing the concatenation of h_i and h_j. The verification sequence for a leaf node consists of the hash values of the sibling nodes on the path from the leaf node to the root. With a leaf node and its verification sequence, the receiver can compute the tree's root value. For instance, $(h_4, h_{1,2}, h_{5,8})$ represents the

verification sequence of packet s_3, as shown in figure 2-1. In [20], Wong and Lam utilize a Merkle hash tree to amortize a digital signature over *n* packets. First, the sender hashes each packet and treats each as a leaf node to the Merkle hash tree. Then, the sender augments each packet with verification information which comprises of the signed root hash value and the corresponding verification sequence of the leaf node. With this knowledge, the receiver can individually and independently verify each packet. This scheme also tolerates packet losses, but logarithmic communication overhead per packet exists since the verification sequence size grows logarithmically with the amount of leaf nodes. In contrast, our proposed scheme, achieves constant per-packet communication overhead.

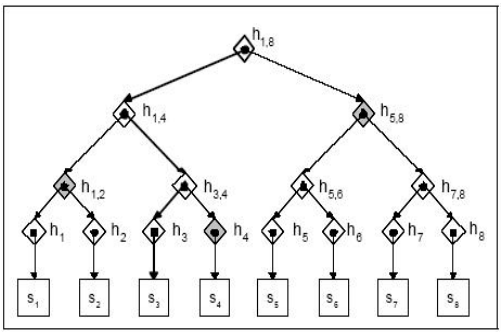

Figure 2-1 Merkle Hash Tree.

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, $h_{1,2}$, $h_{5,8}$).

Erasure codes. An erasure code [7][8][9][12] consists of an encoder and decoder that uses forward error correction to tolerate data loss. The encoder redundantly encodes information into a set of segments. If the decoder receives sufficient segments, it can reconstruct the original information. For example, an *(n, t)* erasure encoder generates a set *S* of *n* symbols $(s_1, s_2, \ldots, s_n)$ from the data. The erasure decoder can tolerate a loss of up to *t* packets. In the next section we detail a signature amortization scheme that employs the use of erasure codes.

2.2. SAIDA

Park et al. in [10][11] propose a signature amortization scheme for multicast authentication, SAIDA (Signature Amortization using the Information Dispersal Algorithm), which utilizes erasure codes to tolerate random packet loss. In this scheme, the sender first partitions the packet stream into blocks of *n* consecutive packets. Next, the sender concatenates the hash values of all packets in one block to form H_j, which it protects by generating a signature $Sign(h(H_j))$ for the hash value H_j. Then, the erasure encoder encodes the verification information *VI*, which includes H_j and $Sign(h(H_j))$, and appends the outputted segments to each packet in the block. Despite the lost of packets, an erasure decoder can successfully reconstruct *VI* as long as it receives a sufficient number of packets. The receiver can then verify H_j with $Sign(h(H_j))$; consequently, the hash values contained in H_j can authenticate all received packets in that block.

Pollution attacks in SAIDA. During normal operation, the receiver validates a block of packets by using erasure codes to decode the verification information from the received packets. Injecting forged packets into the communication channel confuses the receiver, causing it to decode an incorrect H_j. Thus, the receiver is unable to successfully verify the signature $Sign(h(H_j))$ and must drop the received packets. If the receiver attempts to reconstruct the correct information during the pollution attack, it will expend excessive computational power evaluating all possible combinations of the received packets.

2.3. Distillation codes

Karlof et al. [1] propose distillation codes as a mechanism to defend against pollution attacks on a SAIDA-based signature amortization scheme by utilizing Merkle hash trees and one-way accumulators. First, the sender constructs a Merkle hash tree using the hash value of the multicast packets. Next, for each packet, the sender generates and appends a witness, the verification sequence of a leaf node in the Merkle hash tree. When the receiver accepts packets, it separates them into many groups according to each packet's witness. Distillation codes ensure that a group contains all valid packets, allowing the receiver to successfully reconstruct the verification information of this group.

Distillation codes consist of a distillation encoder and a decoder procedure. The distillation encoder first hashes the packets of a single block and concatenates them to form H_j. Then, the distillation encoder uses erasure codes to encode H_j and outputs the set of symbols $S'=(s_1', s_2', \ldots, s_n')$, with each symbol denoting a leaf node from which to build a Merkle hash tree. It also produces a set of distillation code symbols $S=(s_1, s_2, \ldots, s_n)$, where s_i is the concatenation of s_i' and the verification sequence of s_i'. Before multicasting the packet, the sender augments each packet with its corresponding distillation code symbol s_i. Since the receiver can calculate the root hash value of the Merkle hash tree through the verification sequence, the receiver is able to partition the received packet by the calculated root value from the witness of the packet. Valid witnesses possess the same root hash value; thus, the receiver partitions packets with valid witnesses into the same group. As a result, the receiver can reconstruct correct verification information from the packets in the group. Therefore, distillation codes can defeat pollution attacks.

Unfortunately, distillation codes induce logarithmic communication overhead since the witness size grows logarithmically with the number of packets per block. In addition, the receiver will consume significant computation power from erasure decoding and signature verification while suffering a pollution attack. Furthermore, the receiver must buffer all packets, regardless of whether it is valid or invalid, until the correct information is reconstructed, because it does not know the root of the Merkle hash tree in advance. Thus, the receiver requires a large buffer to temporarily store these packets.

There is an additional weakness in distillation codes. An attacker can construct his own Merkle hash tree and transmit packets augmented with the corresponding witness. The receiver will partition these packets into the same group since the witnesses are constructed from the same Merkle hash tree. Because the receiver is unaware of the correct root of the Merkle hash tree beforehand, and an attacker is able to inject a large number of forged packets into one group to exhaust the receiver's computational power, dramatically downgrading the receiver's performance.

3. Proposed scheme

We propose a lightweight and pollution attack resistant multicast authentication protocol (PARM) based upon SAIDA. In our scheme, each packet is appended an evidence containing the verification information that allows the receiver to judge the validity of the packet. PARM is fast and lightweight, which caters to time-sensitive multicast applications and computationally limited devices.

3.1. PARM

Our proposed scheme consists of four phases: initialization, evidence generation, evidence validation, and temporal key renewal. We describe the four phases in the next few sub-sections.

3.1.1. Initialization Phase

In this phase, we define how to generate a temporal key pair, which contains a temporal secret key (TSK) chain and a temporal public key (TPK), using a one-way hash function. The sender creates the evidence of a packet from a TSK chain, and the receiver validates the evidence of a received packet with the TPK.

Before communicating with receivers, the sender must generate the TSK chain and TPK in advance. First, the sender generates k n-bit random numbers $(R_0, R_1,\ldots, R_{k-1})$ and denotes this set of numbers as TSK_0 of the TSK chain. Then, the sender uses the one-way hash function h to recursively generate the remaining TSKs of the TSK chain. By applying the hash function to each member of the previous TSK, the sender can produce the next TSK. For example, TSK_1 is generated by hashing each element in TSK_0 i.e. $TSK_1=(h(R_0), h(R_1),\ldots, h(R_{k-1}))$. The TSK chain has a length of L and is represented as $(TSK_0, TSK_1,\ldots, TSK_{L-1})$. The temporal public key (TPK) is created by hashing every element of TSK_{L-1}.

Figure 3-1 depicts the procedure for TSK and TPK generation. R_0 denotes the randomly generated number, and the arrows specify the direction of the one-way hash function h. Thus, $h(R_0)$ is the hash result of R_0, and $h^2(R_0)$ is the hash result of $h(R_0)$. The set of the elements in the same row comprises a TSK elements array, e.g. $TSK_0=(R_0, R_1,\ldots, R_{k-1})$ and $TSK_1=(h(R_0), h(R_1),\ldots, h(R_{k-1}))$. The elements of the last row form the TPK.

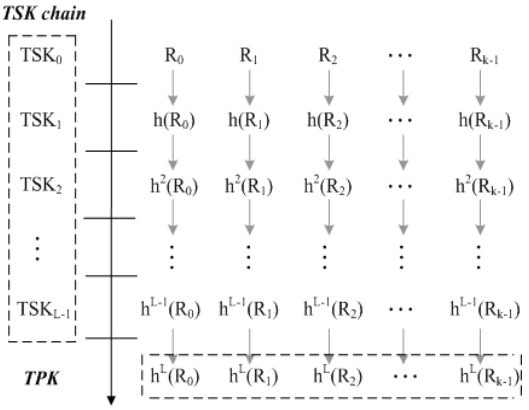

Figure 3-1 Temporal key pair generation.

After successful generation of the TSK chain and TPK, the sender provides receivers with the TPK. Since receivers will use the TPK to determine the validity of received packets, it is vital that the sender sign the TPK with a digital signature to protect it during distribution. Otherwise, an attacker can convince receivers to accept a forged TPK; consequently, all valid packets will fail to pass evidence validation. The receiver stores the TPK if it verifies the signature.

3.1.2. Evidence Generation Phase

Prior to broadcasting a message, the sender must generate for each packet the evidence, or verification information, which allows receivers to determine the validity of a packet. Since each packet is augmented with evidence, the evidence generation phase should be lightweight and fast. For a given temporal key pair, the sender needs to maintain a usage table, such as the one in Figure 3-2, that tracks the number of times each column index of the TSK elements array is used. The row *index* denotes the column index of the TSK elements array, while the row *usage* tracks the number of uses of the corresponding index.

Index	0	1	2	3	4	⋯	k-2	k-1
Usage times	1	5	3	8	0	⋯	0	2

Figure 3-2 Usage table.

Figure 3-3 illustrates the evidence generation phase. To generate evidence E_M for a packet M, the sender first hashes the packet with a one-way hash function h. The hash value is divided into a set of p segments, denoted $S=(i_0, i_1,\ldots, i_{p-1})$, where each segment size is b-bits. Interpreted as an integer between 0 and 2^b-1, each segment in the set S represents a column index of the TSK elements array. For each index i, the sender determines the TSK based upon the usage of i by selecting $TSK_{(L-1)-a_i}$, where a_i denotes the usage of i. Thus, the sender chooses the last TSK of the chain, TSK_{L-1}, if i has never been used. Once the sender determines the TSK, it chooses the i-th element of the selected TSK. For example, if i_0 used $L-1$ TSK elements, then the sender chooses the i_0-th element of TSK_0, which is R_0. Since each segment of S corresponds to an index of the TSK elements array, the sender produces p elements, which constitutes the evidence of the packet. After appending the evidence to the packet, the sender can finally broadcast the packet to the receiver.

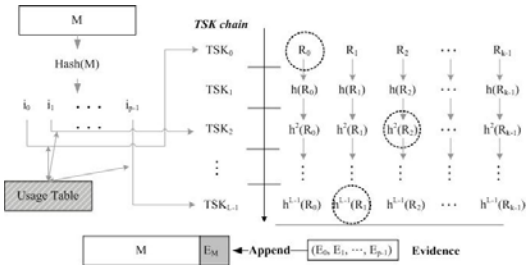

Figure 3-3 Evidence generation phase

3.1.3. Evidence Validation Phase

Upon receiving a packet, the receiver can use the TPK to immediately check the validity of the attached evidence. To forge a packet, the attacker must generate proper evidence for a packet, which is difficult without knowledge of the TSK chain. In section 5, we will demonstrate the complexity of a successful attack. As with the sender, the receiver must also maintain a usage table for each column index of the TSK elements array based on received packets.

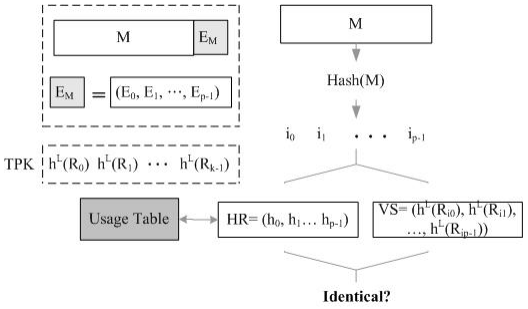

Figure 3-4 Evidence validation phase.

The procedure of the evidence validation phase, depicted in figure 3-4, is similar to that of the evidence generation phase. After receiving a packet containing evidence E_M, the receiver separates the evidence, denoted $E_M=(e_0, e_1,\ldots, e_{p-1})$, from the packet M. To validate the evidence for this packet, the receiver hashes M with the one-way hash function h, which is identical to the one-way hash function used by the sender in the evidence generation phase. Next, the receiver divides the hash value $h(M)$ into p b-bit segments, denoting these segments as the set $(i_0, i_1,\ldots, i_{p-1})$. By interpreting each segment as an integer between 0 and 2^b-1, each segment can represent a column index of the TSK elements array. Each index i, along with its usage a_i, determines the number of times to hash the corresponding element e_i of the evidence. Given an index and its usage, the receiver should perform a_i+1 hashes on the corresponding element of the evidence. Thus, if index i has never been used before, the receiver need only hash e_i once. The ensuing set of hash results from every element of the evidence is denoted by $HR=(h_0, h_1,\ldots, h_{p-1})$. The receiver selects the verification subset $VS=(h^L(R_{i0}), h^L(R_{i1}),\ldots, h^L(R_{ip-1}))$ from

the TPK, where $h^L(R_i)$ is the *i-th* element of the TPK. The receiver considers the evidence valid if the two sets, *HR* and *VS*, contain identical elements, accepting the packet with valid evidence and dropping it otherwise.

3.1.4. Temporal key renewal phase

In the previous three phases of PARM, the sender uses the TSK chain to generate the evidence of a packet, which the receiver validates with the TPK. Since the packet's evidence prevents the receiver from accepting forged packets, our proposed scheme can thwart pollution attacks. Nevertheless, an attacker can still sniff various bits of the TSK chain because each piece of evidence contains elements of the TSK chain. When an attacker obtains enough portions of the TSK chain, the probability of forging valid evidence rises dramatically. Thus, periodic renewal of used TSK elements is necessary to ensure secure communications between the sender and its receivers.

We define a threshold value *T* in our key renewal phase. U_{TSK0} represents the number of used elements in TSK_0 (the first TSK of the TSK chain) since the last temporal key renewal, and the set $(j_0, j_1,\ldots, j_{t-1})$ denotes the indexes of the used elements. When U_{TSK0} exceeds the threshold *T*, new elements are required. First, the sender generates U_{TSK0} new random numbers for the used indexes of TSK_0. Using these random numbers, the sender creates the partial TSK and the partial TPK with the one-way hash function *h* by following the temporal key generation procedure of the initialization phase. The sender then updates its copy of the TSK chain with the partial TSK elements. Since the receiver must also update its TPK, the sender concatenates the new partial TPK with its digital signature *Sign(Partial TPK)*, which it then encodes with erasure codes and appends to outgoing packets. Figure 3-5 illustrates the preparation required for sending the partial TPK to the receiver. Upon successful renewal of the TSK chain and TPK, the sender and receiver may resume evidence generation and verification of packets.

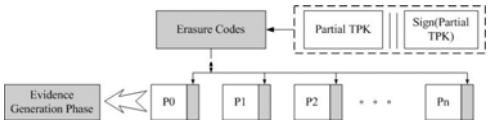

Figure 3-5 Temporal key renewal phase.

3.2. Practical considerations

In this section, we describe the features of our proposed scheme.

Efficient evidence generation and validation. Instead of complicated cryptographic algorithms, PARM utilizes a simple one-way hash function to generate and validate evidence. Because of its lightweight operation, PARM is suitable for devices with limited computational power.

Instant validation. Upon arrival, the receiver is able to validate each packet based on the evidence appended to the packet, allowing it to immediately discard invalid packets. Instant validation also mitigates the attacker's ability to overwhelm the receiver's storage space by sending large amounts of packets.

Packet loss tolerant and individual validation. In some mediums, such as the internet, packet loss occurs frequently. Since some multicast applications may not retransmit lost packets, we design our scheme to tolerate the loss of packets. Moreover, packet loss will not affect the validation of other packets because the receiver independently verifies each packet.

Constant verification information size. To provide for individual packet validation, the sender augments every packet will its own evidence. If the evidence size grows too large, the overhead will significantly affect

performance. In our proposed scheme, the evidence size remains constant per packet.

3.3. Attack resistance

There are various types of attacks that a robust multicast authentication scheme should be able to defend against. In this section, we describe how PARM resists common attacks. We assume it is infeasible for an adversary to successfully forge a packet's evidence, and we detail the degree of difficulty of violating this assumption in section 5.

Injection attack. An attacker injects random or pre-designed packets with the intent of inducing the receiver into performing illegal behavior. Because PARM requires that each packet provide evidence, the receiver simply rejects the injected packet as soon as the evidence validation fails.

Modification attack. Due to the distributed nature of a multicast environment, an adversary may capture a sender's packets, modifying its contents before retransmission to the receivers. As with injection, the receiver will attempt to verify the modified packet and drop it after an unsuccessful validation of the evidence.

Signature flooding attack. Most authentication mechanisms require additional verification information that allows a receiver to validate the received packets. If packet validation entails high operational overhead, an attacker may send a large amount of packets with invalid verification information in an attempt to exhaust the computational resources of a receiver. Because of its lightweight validation procedure, PARM is resistant to signature flooding attacks.

Pollution attack. In this attack, an adversary injects forged packets to pollute the erasure decoding procedure. Since it is impractical for the adversary to generate legitimate evidence for a forged packet, the receiver refuses to accept the packet after evidence validation failure.

4. Comparison

In this section, we compare PARM with distillation codes. Karlof et al. [1] proposed distillation codes as a means to thwart pollution attacks against SAIDA. However, distillation codes require significant communication and computation overhead. Before we begin, we define several parameters in Table 4-1.

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 PARM
N_A	number of attack packets in one block
D_G	number of partitions for distillation codes

Table 4-1 Parameters.

4.1. Storage overhead

In the initial stage, distillation codes require no additional storage size, while PARM needs extra storage at both the sender and receiver. In our proposed scheme, the length of the TSK chain is L, and each TSK contains k elements. Thus, the sender's TSK elements array contains $L*k$ elements, while the receiver's TPK has a size of k elements. A long TSK chain requires a large amount of storage; however, the sender of a multicast environment typically possesses the resources to cope with this

overhead

During a pollution attack, PARM saves considerable storage space over distillation codes. Since our proposed scheme instantly checks the validity of received packets, the receiver only buffers valid packets. In contrast, distillation codes cannot immediately judge the validity of received packets. Consequently, the receiver is forced to buffer all packets, regardless of its validity. Because the typical receiver has limited resources, buffering many packets degrades the receiver's performance. Therefore, PARM is more space efficient on the receiver end during a pollution attack than distillation codes.

4.2. Communication overhead

Both distillation codes and PARM append validation information to a packet. Because distillation codes utilize Merkle hash trees, the witness of a packet grows logarithmically with the number of packets per SAIDA block. On the contrary, our proposed scheme employs constant sized evidence for any number of packets per block. Therefore, the communication overhead of PARM scales better than that of distillation codes.

4.3. Computation overhead

Likewise, the computational overhead of the receiver grows logarithmically in distillation codes while our scheme's overhead remains constant. We first show the computational overhead required by the sender or receiver to send or receive one block of packets, respectively, while operating normally. For distillation codes, the sender's cost of transmitting one block of packets is

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

while the receiver requires a computational overhead of

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

to validate one block of packets. In contrast PARM only needs a computational overhead of

$$N_K * C_H + S_G$$

by the sender to prepare a block of packets for broadcast, while the receiver's cost of validating a block of packets is

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

Because a pollution attack does not affect the computational overhead of the sender, we focus only on the analysis of the receiver.

During a pollution attack, the use of distillation codes requires a cost of

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

Compared to our proposed scheme, the receiver's overhead is

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

Erasure codes and signature verification need significant computational power. Unfortunately, distillation codes require even more calculations than either of these two computationally consuming operations. Thus, PARM is more lightweight than distillation codes.

5. Security analysis

In this section, we analyze the degree of difficulty of computing valid evidence for a packet without prior knowledge of the full TSK chain.

For the first case, we assume the attacker does not possess any element of the TSK. On average, an adversary must guess $(2^{b-1})^p$ hash values, where *b* represents the number of bits per element and *p* represents the number of elements in each evidence. The complexity of finding valid evidence for a packet can be denoted as $O(2^{bp})$.

In the second case, we derive the probability P_f of the adversary producing valid evidence for a packet

given n TSK elements. Since each element in the TSK will not be reused, except those elements in the first TSK chain, TSK_0, we can assume that only the reused elements of TSK_0 will affect the security of PARM. The TSK has a chain length of L and k elements per TSK. If the known elements are distributed among each TSK, then n/L denotes the number of these elements in TSK_0. Without executing the temporal key renewal phase, we derive a tight upper bound for P_f.

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

We define the security strength as

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

Equation 5.2 shows that the security strength S increases in direct proportion to the number of TSK elements k or TSK chain length L and decreases with the number of used TSK elements n. The security strength S decreases if probability P_f increases too much.

In the final case, we execute the temporal key renewal phase, rendering useless any evidence obtained by the attacker. With T denoting the threshold value as defined in the temporal key renewal phase, the upper bound probability of forging evidence is

$$P_f = \left(\frac{\frac{n}{L}}{k}\right)^p \quad \frac{n}{L} \le T \quad (Equation\ 5.3)$$

and the security strength can be represented as

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

Since n/L denotes the number of used elements in TSK_0, we reset n/L to zero if it exceeds the threshold T.

6. Evaluation

In this section, we evaluate the security strength of our proposed scheme under different conditions. By adjusting the parameters of the equations derived in section 5, we can assess the security strength of PARM under various situations.

Utilizing equation 5.2, we first discuss the security strength of our scheme without any key renewal. Figure 6-1 and Figure 6-2 illustrate the change in security strength as a result of varying the evidence size and the number of TSK elements an attacker already possesses. We apply the following parameters to equation 5.2: each TSK contains k=512 symbols, each TSK chain length is L=10, and each piece of evidence contains p TSK elements. For a small number of TSK elements known to the adversary, Figure 6-1 demonstrates that security strength greatly increases as the evidence size increases. Figure 6-2 depicts a significant decrease in security strength when the adversary possesses a much larger collection of TSK elements.

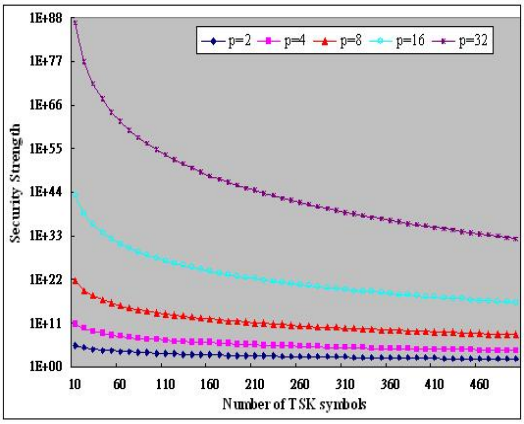

Figure 6-1 The security strength of different evidence size.

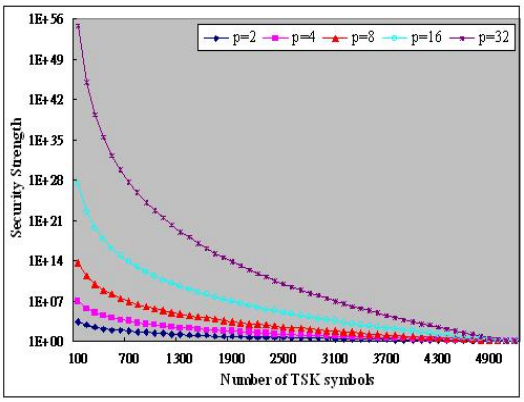

Figure 6-2 The security strength of different evidence size.

TSK chain length *L* also influences the security strength of PARM. Applying equation 5.2, we set the evidence size *p* at 16 and the number of elements *k* per TSK at 512. From Figure 6-3, we observe that longer TSK chains increase the security strength. Since we do not employ the key renewal mechanism in this test, the security strength significantly dramatically drops as the adversary obtains more TSK elements.

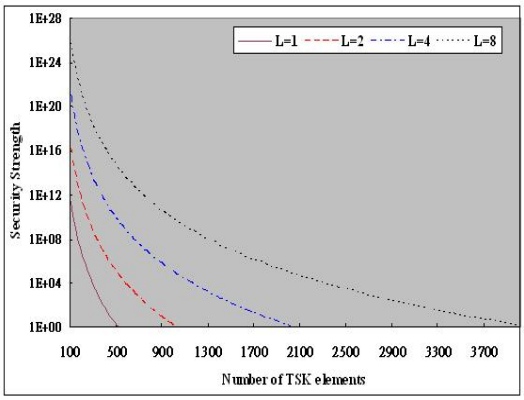

Figure 6-3 The security strength of different TSK chain length.

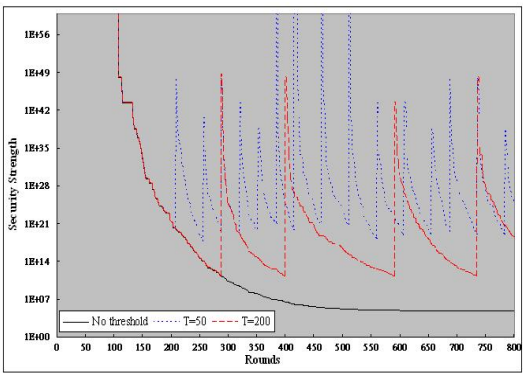

Figure 6-4 The security strength with key renewal.

Equation 5.4 models the security strength of our scheme with the temporal key renewal mechanism in effect. The key renewal phase occurs when the number of used TSK elements in TSK_0 reaches a threshold T, forcing a partial renewal of the temporal key pair. Figure 6-4, depicts the security strength of PARM under different key renewal thresholds *T*. The x-axis represents the number of transmission rounds between sender and receivers. Because the first curve does not renew its key pair, its security strength monotonically decreases. In contrast, the other curves do renew their keys, and thus can maintain a minimum level of security. For example, a threshold of 200 induces a minimum security strength of *E+11*, while the security strength remains above *E+19* for a threshold of 50. From the figure, we can conclude that a small threshold can sustain higher security strength than a large threshold; therefore, key renewal is essential to guarantee security.

7. Conclusion

Pollution attack is a significant problem in multicast authentication. Despite past efforts, researchers have not been able to develop an efficient solution. This paper proposes a new approach to resisting pollution attack that not only offers the sender and receiver with lightweight computational overhead but also allows the receiver to instantly validate packets without the need to buffer invalid packets. The partial key renewal mechanism provides a guarantee on a lower bound of the security regardless of the amount of disclosed TSK elements. In addition to SAIDA, other signature amortization schemes that rely on fault-tolerant algorithms to defend against pollution attacks could benefit from our proposed approach.

We provide an analysis on the storage, communication, and computational overhead which demonstrates that our proposed scheme is relatively lightweight to previous solutions. We also evaluate our scheme under various conditions to help senders define operating parameters suitable for its local network. Because the storage size of the TSK elements array is considerable, we aim to reduce the storage overhead in future work.

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, 199**7.**