

國立交通大學

資訊工程學系

碩士論文

A Pollution-Attack Resistant Multicast Authentication Scheme

Tolerant to Packet Loss

研 究 生：林渥人

指導教授：謝續平 博士

中華民國九十五年六月

A Pollution Attack Resistant Multicast Authentication
Scheme Tolerant to Packet Loss

研 究 生：林渥人

Student: Warren 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



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 2006
Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 五 年 六 月

A Pollution Attack Resistant Multicast Authentication Scheme Tolerant to Packet Loss

Student: Warren Lin

Advisor: Shih-Pyng Shieh

Department of Computer Science and Information Engineering

National Chiao Tung University

Abstract

Assuring authenticity of packets is a critical security measure in multicast applications. Due to the high overhead of signing every multicast packet with a digital signature, schemes employing signature amortization abate this cost by endorsing a block of packets at once. By utilizing a fault-tolerant coding algorithm, signature amortization schemes can tolerate packet loss. However, enhancing these schemes with a fault-tolerant coding algorithm introduces pollution attacks, a form of denial of service attack in which the adversary injects invalid symbols into the decoding process. Unfortunately, previous solutions that combat pollution attack required time synchronization or were computationally inefficient. To address these problems, we propose a multicast authentication scheme resistant to pollution attack that eases the time synchronization restriction and validates packets with significantly less hash computations.

This is my thank you to all of you. I'll try not to be too poignant.

謝續平老師, thank you for making room in your lab for me and for your guidance, support, and constructive criticism during my graduate study. Also, thank you for putting together a great group of people. 楊明豪學長, thank you for getting me started with a variety of topics and problems and for working with me at Cisco my first year. I know it was difficult for you as my Chinese was horrible in the beginning. 黃育綸學姐, thank you for being so kind and easy to chat with at the conference.

助理, thank you for all for shooting the breeze when I needed a break from studying and for being easy on the eyes. Suki, thanks for being so gullible. Just kidding! Seriously, thanks for the small things like 曼秀雷敦 for mosquito bites or 粽子 for 端午節. Angel, thanks for trading forehands with me in tennis and smashing the birdie at me in badminton. 筱倫, thanks for being nice, but don't let Angel 教壞你; you are already starting to 騙人!

博士班學長姐, thank you all for doling out advice when I needed it. 瘦瘦的士一學長, thanks for introducing me to 臭豆腐. I will never forget that experience, taking refuge at the 系計中 with ALee. 有車的子逸學長, thanks for leading the lab this year and for having the coolest gadgets. 可愛的佳純學姐, thanks for bringing 台中's famous 雞爪 on our trip. Flavored skin and bones – yummy. 快快的羅堯學長, thank you for taking me under your wing and for opening my eyes to things I had never really given much thought to before.

碩士學長, I'm sorry you guys weren't around much this year. I wish I could have gotten to know you guys better. 阿熹學長, thanks for representing 交大 with Wretch and for kicking butt in 躲避球 last year. 櫻皇學長, thanks for being even more quiet than me.

碩二同學, thank you all for taking care of me the last two years; you guys are the best! ALee, thanks for showing me the ropes at the gym, for tossing the pigskin anywhere anytime, for watching Jack Bauer take out one day's worth of bad guys, and for infusing my blood with a need for speed. Jelly, thanks for being my left side and right side neighbor and for catching up on the previous night's game the next morning, be it basketball, soccer, baseball, anything at all. Also, your penchant for a quick game of Ages or NBA Live makes me feel less guilty during breaks. 野狗, thanks for having a stomach that just can't seem to become full, for dominating the

opposition in tennis, for possessing a 美眉 radar with incredible range, for declaring independence at the most random moments, and for protecting Taiwan's environment. 大的, thanks for having unrivaled facial hair growth, for exhibiting lightning speed in Tetris, and for showing us how it's done at KTV. I expect an advance copy of your first CD and a backstage pass to your concert.

碩一學弟妹, thank you all for keeping the lab lively this year; you guys are a colorful bunch! 阿波, it starts with you! Thanks for cracking a joke on just about anything. 紀樹, thanks for wanting to shoot hoops almost every waking moment. 之涯, thanks for bringing the flavor of Taiwan everywhere by wearing flip flops! Too bad you didn't wear them to ASIACCS though. 浩洋, thanks for towering over everyone and for kicking some serious butt in badminton. 朝彥, thank you for being a friend right from the start. You've helped me get through some tough times this year. And last, but not least, 仁倩, thank you for always being so sweet. Your smile and laugh lifts me up on days I'm feeling down. I think it's the little things in life that we live for.

While living in a different, yet somewhat familiar environment these last two years, I feel like I have done some good things and I have made many mistakes, both academically and personally. But as the saying goes, "we only learn from our mistakes." Through it all, I feel like I have taken a step forward and I have grown a little more. I am grateful to have known all of you. I have learned so much from each and every one of you. You all have made me feel comfortable here, enough so that I can finally regard Taiwan, my birthplace, as my home as well. I love you guys and girls.

Sincerely,

渥人
Warren

Table of contents

1. Introduction.....	1
2. Related work	5
2.1. Signature amortization	5
2.1.1. Hash Graphs.....	5
2.1.2. Merkle Hash Trees	6
2.1.3. Erasure Codes	6
2.2. Pollution Attack	6
2.2.1. Distillation Codes.....	7
2.2.2. One-Way Hash Chains.....	7
3. Proposed scheme.....	9
3.1. Features	9
3.2. Assumptions.....	10
3.3. Proposed Scheme	10
3.3.1. Initialization Phase.....	11
3.3.2. Generation Phase	15
3.3.3. Validation Phase.....	18
3.3.4. Key Renewal Phase.....	22
4. Evaluation	23
4.1. Protocol Analysis	23
4.1.1. Packet Authenticity	23
4.1.2. Packet Loss Robustness	24
4.1.3. Packet Replay Rejection	24
4.1.4. Packet Reordering Robustness.....	24
4.1.5. Denial of Service Resistance	25
4.2. Overhead Comparison	25
4.2.1. Computation.....	25
4.2.2. Communication.....	26
4.2.3. Storage	26
5. Conclusion	28
References.....	29

List of Figures

Figure 3-1. Temporal Key Pair Generation.....	13
Figure 3-2. Temporal Key Pair Notation	14
Figure 3-3 Current Elements Table	15
Figure 3-4 Usage Table	15
Figure 3-5. Evidence Generation	17
Figure 3-6. Worst-case Verification	19
Figure 3-7 Optimal Case Verification	21



1. Introduction

Instead of individually transmitting the same data to each receiver, multicast communication permits one or more senders to simultaneously send the same data to many receivers. This communication technique decreases the workload of the sender, greatly raising its efficiency. Applications of multicast communication include system announcement (new keys, network time), monitoring (headlines, stock prices), collaboration (distance learning, multimedia conferencing, multiplayer gaming), and e-commerce (auctions).

Unfortunately, a malicious user may drop, delay, or modify intercepted communication packets or inject their own packets into the data stream. Security measures offer different benefits, such as authentication, authorization, confidentiality, integrity, and non-repudiation. At a minimum, authentication prevents the attacker from hijacking the data stream and deceiving receivers into accepting fabricated data.

Authentication typically employs one of two cryptographic fundamentals: symmetric cryptographic primitives or asymmetric cryptographic primitives. Approaches that use symmetric cryptographic primitives share an identical secret key between the sender and receiver. For example, a sender generates a MAC (Message Authentication Code) as a function of the message and the secret key. The receiver may authenticate the packet by calculating the MAC using the secret key. However, simply employing a MAC for multicast authentication without introducing an asymmetric mechanism, such as delayed key disclosure used in TESLA (Timed Efficient Stream Loss-Tolerant Authentication) [10][15], will allow adversaries to easily forge a MAC by capturing a receiver and using the secret key. Unfortunately, the nature of delayed key disclosure inherently leads to denial of service attacks, as adversaries can overwhelm the receivers' buffers before they obtain the key to verify

the message. In contrast, methods that use asymmetric cryptographic primitives involve a public and private key pair. For instance, the sender generates a digital signature with the private key, which the receiver can verify using the public key. However, generating a digital signature for each message incurs a significant computational and bandwidth cost. A one-time signature is a type of digital signature that can sign a predetermined number of messages. Despite their speed and efficiency over ordinary digital signatures, schemes that utilize one-time signatures [3][7][11] suffer from large key sizes.

Signature amortization defrays the cost of digital signatures by generating a single signature over a block of packets. Receivers authenticate the signature after it has obtained all the packets in the block. Based on a multitude of techniques, early signature amortization schemes offer different benefits. For example, SAIDA (Signature Amortization using the Information Dispersal Algorithm) [4][8] utilizes fault-tolerant encoding to tolerate random packet loss. However, they all suffer from denial of service attacks. Fault-tolerant schemes, in particular, are vulnerable to pollution attacks, a type of denial of service attack in which an adversary disrupts the decoding process by introducing invalid symbols. Previous schemes resistant to pollution attacks, such as PRABS (Pollution Resistant Authenticated Block Streams) [2] and PARM (Pollution Attack Resistant Multicast) [1], require time synchronization between the sender and receiver. Unfortunately, time synchronization may cause a receiver to suffer significant buffer consumption, like in PRABS, or prevent it from recovering from an unsynchronized state, like in PARM.

While devising a multicast authentication scheme, we must take into account the following design requirements:

- *Packet authenticity.* An adversary must not be able to forge his own packets without knowledge of the secret key chains. Receivers must

validate the origin of each packet to ensure the refusal of injected or modified packets.

- *Packet loss robustness.* Individual authentication permits receivers to authenticate packets regardless of burst, correlated, or other deliberate loss pattern of packets.
- *Packet replay rejection.* A receiver must be able to detect and reject replayed packets.
- *Packet reordering robustness.* Independent validation allows receivers to authenticate out-of-order packets.
- *Denial of service resistance.* A lightweight and immediate authentication mechanism allows receivers to withstand denial of service attacks against their storage and computation resources.

In this paper, we propose a multicast authentication scheme that is both lightweight and resistant to pollution attack. By using one-way hash functions, our scheme can quickly generate and verify packets. Since our proposed scheme can immediately and independently authenticate a received packet, it does not risk exceeding buffer space with unverified packets during a pollution attack. Schemes that rely on fault-tolerant coding to provide packet loss tolerance can employ our approach to defend against pollution attacks. We make the following contributions:

- We do not assume time synchronization between sender and receiver. This assumption allows for immediate and individual authentication of received packets; therefore, our scheme can easily recover from disordered or lost packets.
- We introduce a recent key storage mechanism in which the receiver retains a window of the w most current hash values of each hash chain. This mechanism permits the receiver to execute a minimal number of

hashes to validate slightly out-of-order packets.

- We suggest a secure, yet economical, key chain renewal plan. This plan prevents hash value reuse while avoiding unnecessary waste of unused hash values.

In the following section, we discuss related work in signature amortization. We also describe the problem of pollution attack in signature amortization and identify schemes that defend against it. Next, section 3 details our proposed scheme by highlighting our scheme's features, indicating our assumptions, and specifying its various phases. In addition, we provide an analysis of our scheme and a comparison of its performance to PARM in section 4. Finally, we conclude by summarizing our findings in section 5.



2. Related work

We introduce the idea of signature amortization in section 2.1 and describe current research in this area. In section 2.2, we define the problem of pollution attack, which plagues the aforementioned signature amortization schemes. We also present schemes that combat pollution attack and discuss their weaknesses.

2.1. Signature amortization

Due to the considerable computation and communication overhead in digital signature based multicast authentication, signature amortization is utilized to allay these expenses by generating a single digital signature over many packets instead of each packet individually. Signature amortization schemes differ in their implementation and can be classified into several categories: hash graphs, Merkle hash trees, and erasure codes.

2.1.1. Hash Graphs

Approaches that use hash graphs [6][12][14][15] construct a directed acyclic graph where each vertex corresponds to a packet and edges indicate hash direction. In addition, each vertex contains the hash value of the neighbors on its incoming edges. Terminating the hash graph by endorsing it with a digital signature allows a receiver to authenticate a packet in the hash graph by validating the hashes along the path to the signature packet. Unfortunately, signature flooding attacks render hash graphs vulnerable by overwhelming receivers' computational and storage resources. Furthermore, hash graphs suffer from deliberate signature packet loss, which prevents the authentication of any packets.

2.1.2. Merkle Hash Trees

A Merkle hash tree [20] is a binary tree whose leaves consist of the hash of its data blocks. Nodes further up the tree comprise of the hash of the concatenation of their respective children. This family of schemes [16] constructs a Merkle hash tree over a block of packets and utilizes a digital signature to sign the root of the tree. For each packet, the sender appends the verification information, which comprises of the signed root and its authentication path, that is, the nodes in the Merkle hash tree necessary to recreate the root. By including the signed root, the receiver can immediately authenticate the packet. Unfortunately, these schemes also suffer from signature flooding. Moreover, each packet's verification information grows logarithmically as the number of leaf nodes increases.

2.1.3. Erasure Codes

Schemes based on erasure codes [13][17][18][21] encode a message of length n blocks into a set of blocks greater than n . If the receiver obtains a sufficient number of symbols, it can accurately reconstruct the message. The rate r is the fraction of symbols required to rebuild the message. Thus, erasure codes can tolerate a maximum loss of $n - r$ symbols per message. Despite its low overhead and ability to tolerate random packet loss, schemes utilizing erasure codes [4][5][8] are susceptible to deliberate corruption of symbols.

2.2. Pollution Attack

A pollution attack is a type of denial of service attack in which an adversary injects forged symbols into the data stream. When a receiver attempts to reconstruct the message, it expends considerable buffer space and computation power. Two schemes have been proposed to tackle pollution attack in a signature amortization

scheme: PRABS utilizes distillation codes, while PARM employs one-way hash chains.

2.2.1. Distillation Codes

Proposed by Karlof et al. [2], PRABS employs distillation codes to resist pollution attacks by distilling the valid symbols of an erasure encoding from the invalid ones. To achieve this task, the distillation encoder accumulates a set of valid symbols and appends a witness to each symbol. Because distillation codes utilize Merkle hash trees as one-way accumulators, the size of the witness increases logarithmically with the size of the accumulated set. To recover a valid message, the distillation decoder uses a symbol's witness to partition the received symbols such that valid symbols are separate from invalid ones. When a partition collects enough symbols, it can use an erasure decoder to attempt a reconstruction of the message. Unfortunately, the receiver requires a large amount of storage to temporarily buffer received symbols. Since the receiver has no knowledge of the root of the Merkle hash tree in advance, it must retain valid and invalid symbols alike.

2.2.2. One-Way Hash Chains

A one-way hash function [19] is a cryptographic mechanism that can quickly and easily calculate a hash value; however, it is infeasible to compute the original message using the hash value. Given a message M of variable length, its fixed-length hash value is generated by a hash function H of the form $h = H(M)$. A secure hash function must satisfy three critical properties:

- *One-way.* For any given hash value y , it is computationally infeasible to find x such that $h(x) = y$.
- *Weak collision resistance.* For any given value x , it is computationally infeasible to find $y \neq x$ with $h(x) = h(y)$.
- *Strong collision resistance.* It is computationally infeasible to find any

pair (x, y) such that $h(x) = h(y)$.

To create a one-way hash chain, the output of a hash function is repeatedly hashed until achieving the desired length of the chain. Based on a set of one-way hash chains, Lin et al. [1] designed a signature amortization scheme, PARM, which augments each packet with a set of hash values as its verification information. Unlike PRABS, this scheme consumes constant communication overhead by assuming time synchronization between sender and receiver. A lack of synchronization, however, reduces the ability of PARM to instantly validate a received packet or even prevents its recovery to normal operations. For long hash chains, moreover, PARM expends significant computation power during validation as it recursively hashes the verification information.



3. Proposed scheme

In section 3.1, we describe the features of our proposed scheme. Next, we define our assumptions in section 3.2. Finally, section 3.3 details the operations of our scheme's four phases.

3.1. Features

We devise a multicast authentication scheme resistant to pollution attack that remains computationally lightweight through the use of one-way hash functions for its cryptographic primitive. Unlike PRABS, our scheme does not require the receiver to buffer received packets, regardless of its validity, before authentication. While PARM improves upon PRABS because it can immediately validate a packet, it assumes time synchronization between sender and receiver. Since our scheme does not require time synchronization, it can instantly and independently authenticate out-of-order packets. In addition, the lack of time synchronization allows our scheme to recover in the event that packets are lost. Furthermore, our scheme saves the receiver considerable computational power: packets arriving in order cost only one hash computation per segment of evidence while packets arriving just slightly out-of-order require just a few additional hash computations per segment of evidence. Nevertheless, receivers can still validate packets that are significantly out-of-order using the original public key. Finally, our scheme provides a key renewal mechanism that prevents the reuse of any hash values. In addition, we can assure that at least half the length of each key chain will be used in the worst-case scenario.

3.2. Assumptions

We assume the sender is a relatively powerful device that cannot be compromised. A key distribution mechanism is required to securely deliver the public key to all the intended receivers. Moreover, the receivers are guaranteed to obtain the new public key during the key renewal phase.

3.3. Proposed Scheme

In the following sub-sections, we detail the four phases of our scheme: initialization, evidence generation, evidence validation, and key renewal. We summarize our notation in Table 3-1.



Table 3-1. Parameter Definitions

C_i^j	j -th current TSK element of the i -th TSK chain
E_i	i -th Evidence
E	Evidence of a packet
P	Packet
Q	Sequence number of a packet
R_i	i -th Random number
S_i	i -th Segment
U_i	i -th Usage number
b	Size of segment (bits)
k	Number of TSK chains
l	Length of a TSK chain
n	Size of random number / TSK element (bits)
p	Number of segments
q	Number of sequence numbers
w	Number of current TSK elements per TSK chain

3.3.1. Initialization Phase

During the initialization phase, the sender generates a temporal key pair consisting of a temporal public key (TPK) and a set of temporal secret key (TSK) chains. Similar to asymmetric key encryption, the sender uses the TSK chains to generate evidence for a packet, while the receiver validates the evidence with the TPK.

To create a temporal key pair, the sender first randomly generates k n -bit random

numbers $[R_0, R_1, R_2, \dots, R_{k-1}]$. Next, it recursively applies the one-way hash function h to each random number, constructing a set of k hash chains of length l . Thus, the first TSK chain, TSK_0 , contains the values $[R_0, h^1(R_0), h^2(R_0), \dots, h^{l-2}(R_0), h^{l-1}(R_0)]$, while the last TSK chain, TSK_{k-1} , has values of $[R_{k-1}, h^1(R_{k-1}), h^2(R_{k-1}), \dots, h^{l-2}(R_{k-1}), h^{l-1}(R_{k-1})]$. The TPK consists of the l -th hash of each random number, that is, $[h^l(R_0), h^l(R_1), h^l(R_2), \dots, h^l(R_{k-1})]$. Figure 3-1 illustrates the process of generating the temporal public key and the set of temporal secret key chains. In this figure, the arrows indicate the direction of the one-way hash function during the initialization phase. However, we shall spend the key chain in the opposite direction throughout the evidence generation phase.



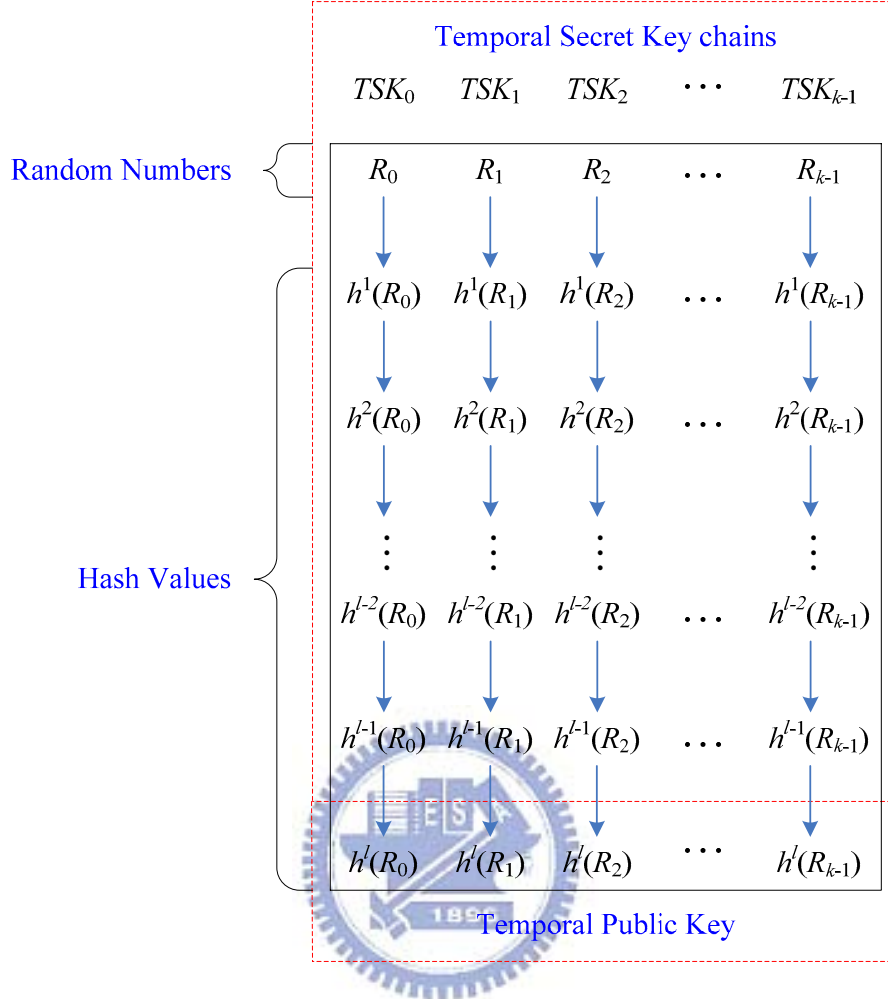


Figure 3-1. Temporal Key Pair Generation

We shall use the notation TSK_x^y to represent individual hash values of the TSK chains, where x and y denotes the TSK chain and the number of recursive hashes required to reach the TPK, respectively. For example, the hash element TSK_{k-1}^l is part of the last TSK chain and must be recursively hashed l times to verify it is equivalent to the TPK. We may also write the last TSK chain as $[TSK_{k-1}^l, TSK_{k-1}^{l-1}, TSK_{k-1}^{l-2}, \dots, TSK_{k-1}^2, TSK_{k-1}^1]$. Correspondingly, we identify individual components of the temporal public key by TPK_x , where x denotes the chain. Therefore, we may represent the TPK as $[TPK_0, TPK_1, TPK_2, \dots, TPK_{k-1}]$.

Error! Reference source not found. summarizes our notation.

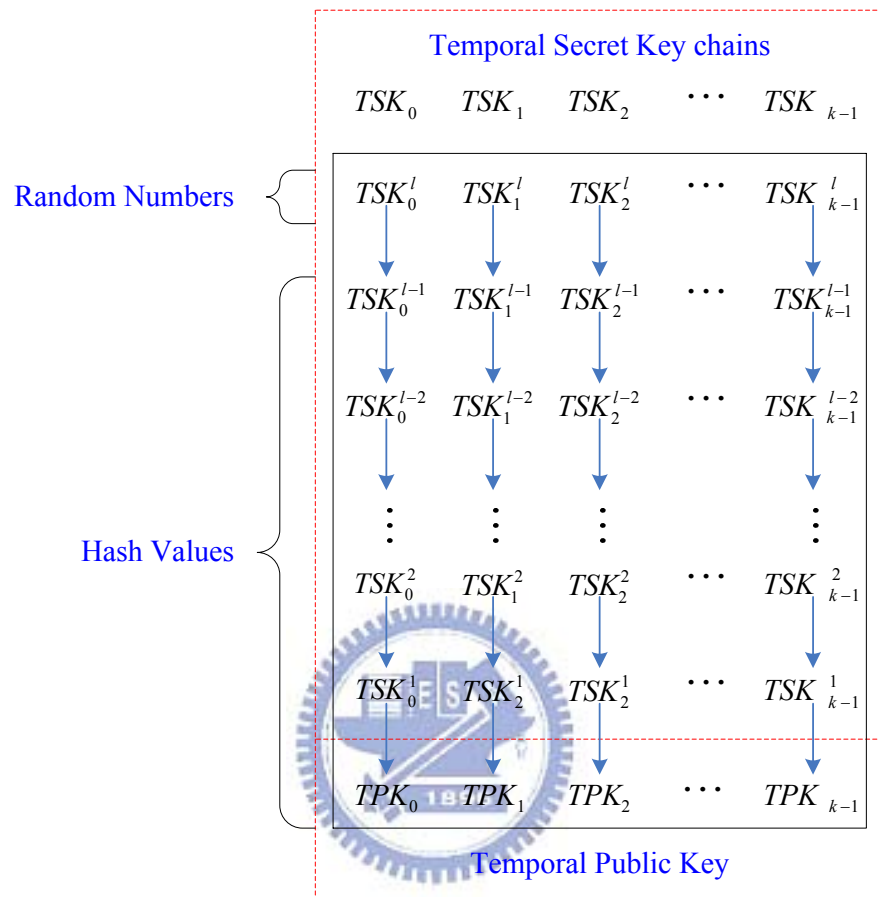



Figure 3-2. Temporal Key Pair Notation

Prior to regular communications with receivers, the sender must securely distribute the temporal public key to all the intended receivers. To demonstrate the authenticity of the TPK, the sender should provide a digital signature or some other means of authentication. In addition to the TPK, the receivers will save a small window of the latest hash values for each TSK chain in the current elements table. **Error! Reference source not found.** provides an example of a current elements table.

Current Elements Table					
<i>Chain Index</i>	TSK_0	TSK_1	TSK_2	...	TSK_{k-1}
<i>Buffer 0</i>	TSK_0^{l-9}	TSK_1^{14}		...	TSK_{k-1}^{l-3}
<i>Buffer 1</i>	TSK_0^{l-8}	TSK_2^5		...	TSK_{k-1}^{l-2}
<i>Buffer 2</i>	TSK_0^{l-7}	TSK_2^6		...	TSK_{k-1}^{l-8}
⋮	⋮	⋮	⋮		⋮
<i>Buffer w-1</i>	TSK_0^{l-5}	TSK_2^9		...	TSK_{k-1}^{l-4}

Figure 3-3 Current Elements Table

The sender will also maintain a copy of the TPK as a reference in the event a receiver requests the TPK. Furthermore, the sender will store the set of temporal secret key chains and use them as a lookup table during the evidence generation phase. Finally, the sender must keep track of a usage table that tallies the number of times each TSK chain has been used. Figure 3-2 depicts a sample usage table.



Usage Table					
<i>Chain Index</i>	TSK_0	TSK_1	TSK_2	...	TSK_{k-1}
<i>Usage Amount</i>	$l-3$	14	0	...	$l-1$

Figure 3-4 Usage Table

3.3.2. Generation Phase

Before multicasting a packet to its receivers, the sender must modify the packet with a sequence number to thwart replay attacks. It must also append evidence, a piece of validation information, which allows the recipient to verify the authenticity of the packet. Because each packet is affixed a unique evidence, the generation phase must only involve lightweight computation.

To generate the evidence E of a packet P and its sequence number Q , the sender utilizes a one-way hash function h to perform a hash. Next, the sender divides the

packet into p segments $[S_0, S_1, S_2, \dots, S_{p-1}]$ of at most b bits in length, where $b = \lceil \lg(k) \rceil$. By interpreting a segment as an integer, the sender can index a specific TSK chain for each segment. Subsequently, the sender consults the usage table to determine the number of times each TSK chain has been used. It will select the next unused hash value for each segment's corresponding TSK chain and append them as the packet's evidence. In addition, the sender will increment and include the usage number of each TSK chain selected as part of the evidence. This permits receivers to immediately validate received packets despite lost or out-of order packets. Lastly, the sender will update its usage table as hash values are expended.



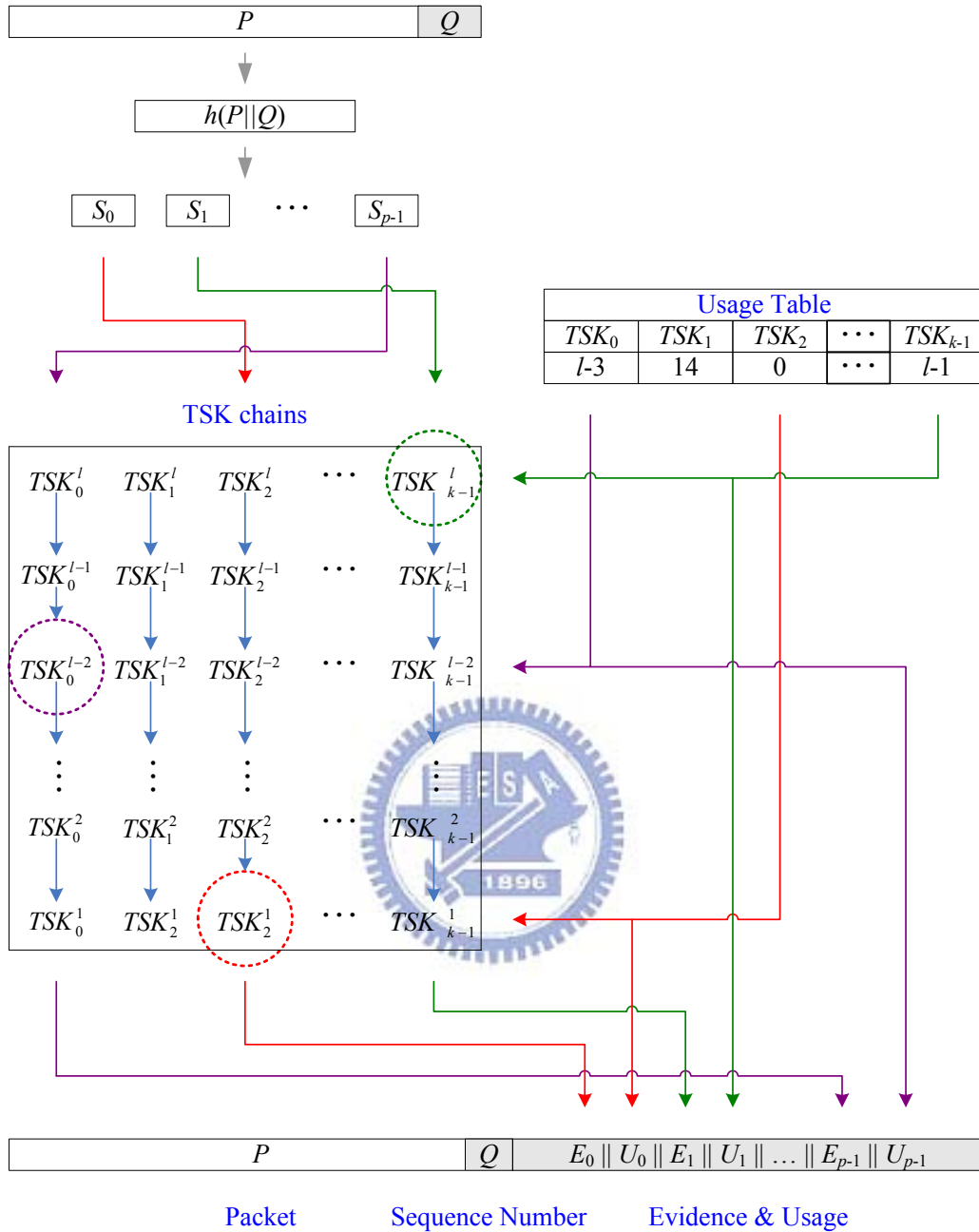


Figure 3-5. Evidence Generation

Figure 3-5 illustrates the operations of the evidence generation phase. For example, the sender hashes a packet P and its sequence number Q by $h(P||Q)$ and then partitions it into p segments. Taken as an integer, segment S_0 corresponds to the third TSK chain. By examining the usage table, the sender can determine that segment S_0 has never been hashed before; thus, it selects element $TSK_2^1 = h^{-1}(R_2)$.

The complete evidence constitutes the corresponding hash value and usage quantity of each of the p segments. After attaching the sequence number and evidence to the packet, the sender can finally distribute the packet to its receivers.

3.3.3. Validation Phase

Upon obtaining a packet, the receiver can authenticate it using the information enclosed in the evidence. In the worst-case scenario, the receiver validates the evidence by recursively hashing each segment of evidence to reach the TPK. However, the optimal case involves a window of current hash values to calculate a minimal number of hashes. The optimal case arises when packets arrive in order (a single hash calculation per segment of evidence) or slightly out-of-order (a small number of hash calculations per segment of evidence).

As depicted in Figure 3-6, the process of verifying a packet's evidence in the worst case is similar to that of generation. To determine the validity of a packet, the receiver must construct and compare two sets of data: the TPK set which involves the public key and the evidence set which involves the hash values contained within the evidence. When a packet arrives, the receiver separates the packet P and its sequence number Q from its evidence E . To build the TPK set, the receiver first hashes the packet with the hash function h . It then splits the outcome into p segments of at most b -bits in length and treats the resultant integers as indexes to the hash chains. Next, the receiver selects the individual components of the TPK corresponding to the index of each segment. For instance, the receiver chooses TPK_2 given segment $S_0 = 2$. When generating the hash set, the receiver hashes each element of the evidence $[E_0, E_1, \dots, E_{p-1}]$ by the corresponding usage amount

$[U_0, U_1, \dots, U_{p-1}]$. Thus, the receiver computes the evidence set by

$[h^{U_0}(E_0), h^{U_1}(E_1), \dots, h^{U_{p-1}}(E_{p-1})]$.

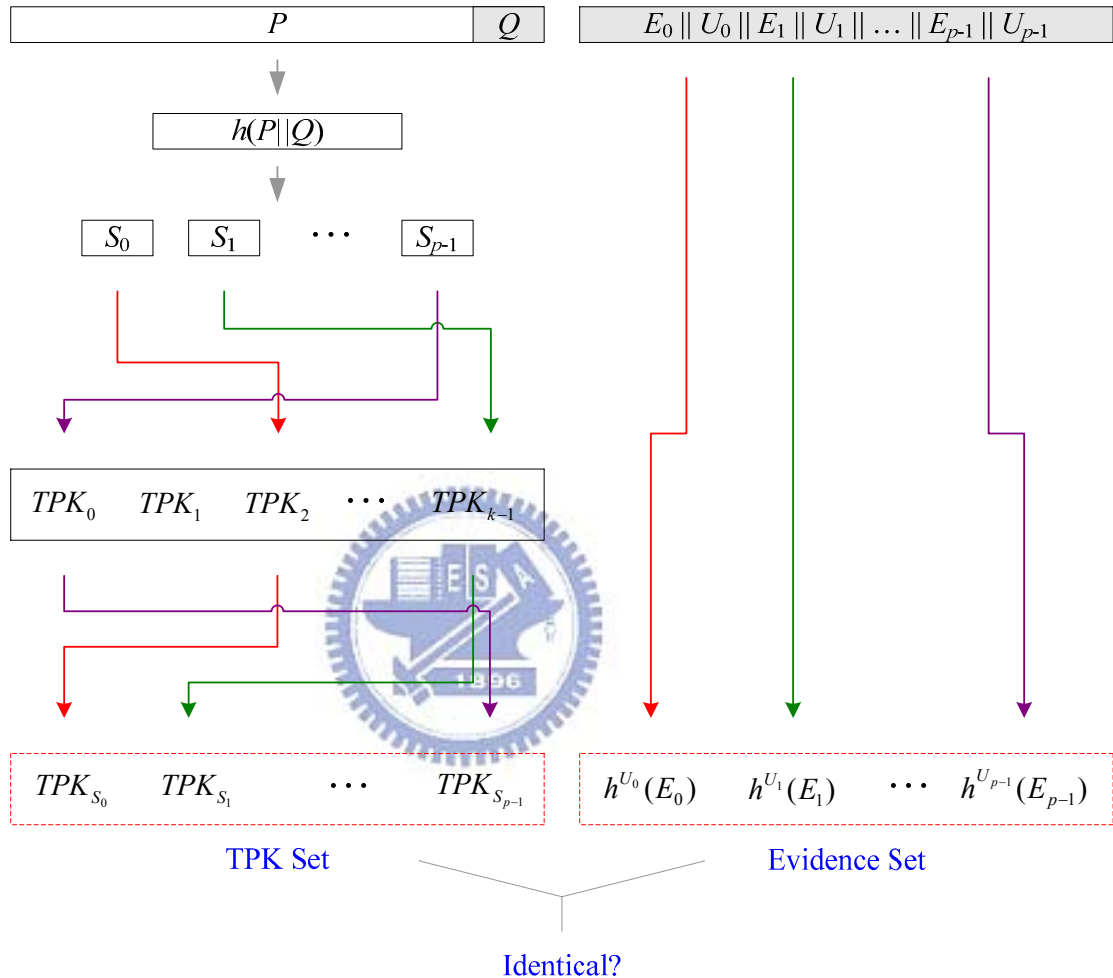


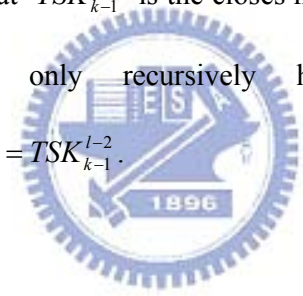
Figure 3-6. Worst-case Verification

Unlike the worst-case scenario, the receiver does not need to recursively hash a segment entirely to the TPK component in the optimal case. Instead, the receiver need only hash a segment to the nearest validated hash value since the receiver buffers the latest accepted w hash values for each hash chain. **Error! Reference source not found.** illustrates the verification of evidence in the optimal case. As described

previously, the receiver validates a packet by separating the packet and sequence number from the evidence. It then hashes the packet using the hash function h . After partitioning $h(P \parallel Q)$ into p segments, the receiver interprets each segment as an integer index to the hash chains. Given a selected hash chain, the receiver checks the current elements table to determine if a previous validated hash value exists in the window. If successful, the receiver hashes the corresponding element of the evidence to the nearest hash value. Let $E = TSK_x^y$, $U = y$, and TSK_x^z be the closes hash value in the buffer, then the receiver computes $h^{y-z}(TSK_x^y)$ to verify E .

For example, given $E_1 = TSK_{k-1}^l$ and $U_1 = l$, the receiver consults the table of current elements and finds that TSK_{k-1}^{l-2} is the closes hash value in the buffer. Thus, the receiver needs to only recursively hash E_l twice, that is,

$$h^{l-(l-2)}(TSK_{k-1}^l) = h^2(TSK_{k-1}^l) = TSK_{k-1}^{l-2}.$$



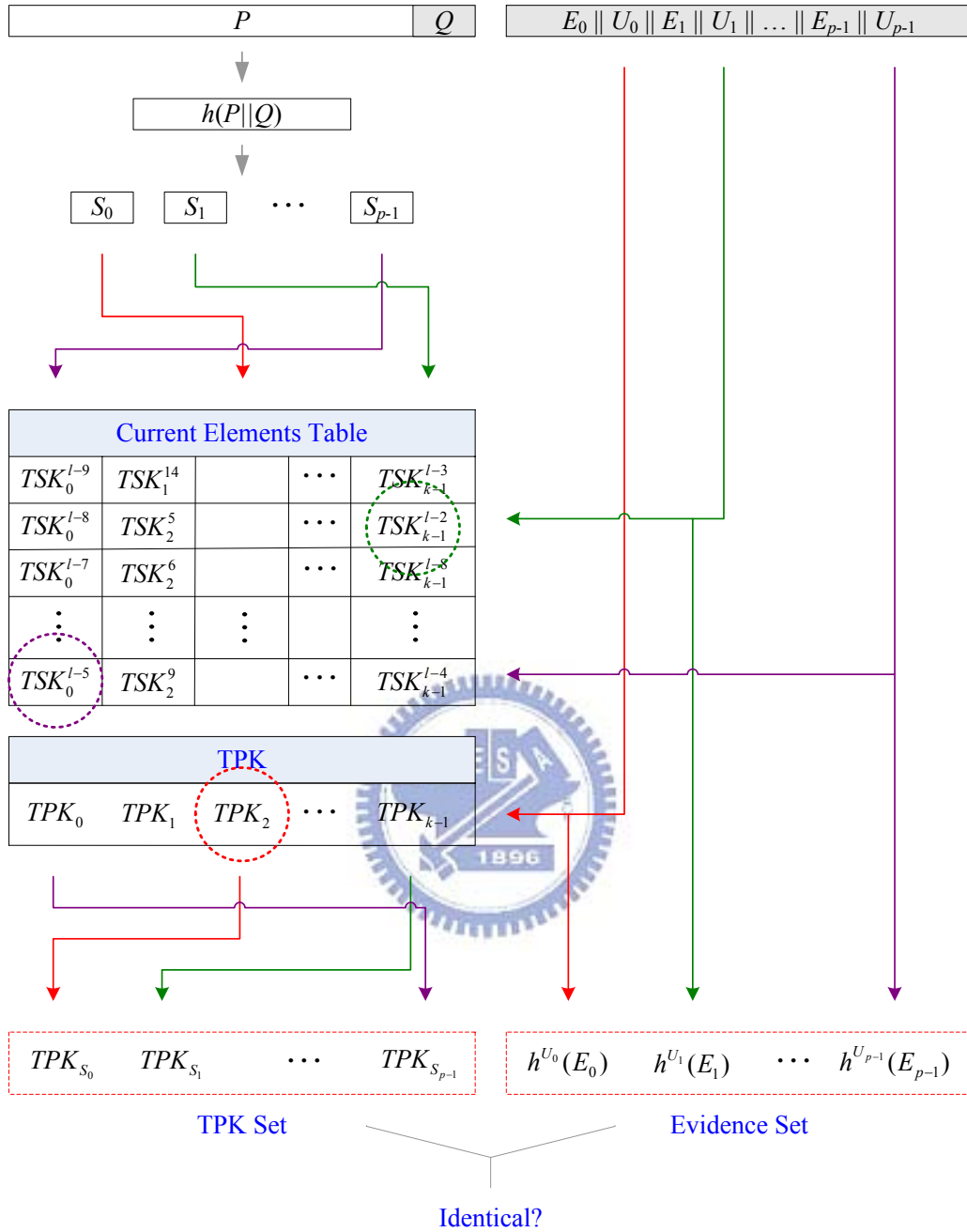


Figure 3-7 Optimal Case Verification

The receiver accepts packets whose segments completely match its corresponding TPK components or latest hash elements. Otherwise, the evidence fails the evidence check, and the receiver immediately rejects the packet.

3.3.4. Key Renewal Phase

Without the TSK chain, it is infeasible for an adversary to augment packets with forged verification information that a receiver will accept. Nevertheless, our scheme must not reuse any hash values of the TSK chain. Once the sender exhausts a key chain, it will search for other chains which have used over half its key space and regenerate new key chains to replace them. Not only does this plan prevent the sender from reusing hash values, but it also strikes a balance between the number of wasted hash values and the number of key chain renewals.



4. Evaluation

Now that we have presented a detailed description of our scheme, we shall provide an analysis per our original requirements in section 4.1 and compare its performance to PARM in section 4.2.

4.1. Protocol Analysis

We analyze the security of our scheme in terms of its ability to authenticate packets, resume from packet loss, reject replayed packets, recover from reordered packets, and resist pollution attacks.

4.1.1. Packet Authenticity

Given any received packet, a receiver can determine the authenticity of the packet. We assume every receiver possesses an authentic copy of a legitimate sender's temporal public key and that h is a collision-resistant hash function. To successfully forge a packet by injection or modification, an adversary must either obtain the set of temporal secret key chains or perform a brute force attack on the hash function to find collisions. In our scheme, only the sender has knowledge of the set of TSK chains. Since the security strength of a hash function against a brute force attack depends solely on the length of the resultant hash value, our scheme provides a strong collision resistance of $2^{p(n/2)}$, where p is the number of hash values per packet and n is the number of bits per hash value. For a hash function to have strong collision resistance, it is computationally infeasible to find any pair (x, y) such that $h(x) = h(y)$. Therefore, it is impractical for an adversary to inject new packets or modify existing ones.

4.1.2. Packet Loss Robustness

Although a network may inadvertently lose packets or an adversary may deliberately drop packets, a receiver can continue to verify other received packets. Because each packet's verification information contains both the hash value and the amount of hash calculations per hash value, a receiver can validate each packet both immediately and independently of other packets. Thus, the loss of packets does not affect the ability of the receiver to check the authenticity of other packets.

4.1.3. Packet Replay Rejection

Because an adversary can collect packets and replay them later, a receiver must be able to identify a duplicate packet and discard it. In our scheme, each packet contains evidence of each packet. Before evidence generation, we append a sequence number to each packet. We then generate the evidence over both the content of the original packet and its sequence number. Therefore, the receiver concurrently validates the sequence number when it checks the authenticity of the packet. Since the receiver keeps track of spent sequence numbers, it can expose replayed packets.

4.1.4. Packet Reordering Robustness

Due to the exposed nature of networks, an adversary can intercept and reorder packets before delivering them to the receiver. The receiver must retain the ability to immediately validate any packet even though it receives packets out-of-order. With the exception of the TPK, which we assume the receiver possesses, a packet's verification information is self-contained. Based upon the contents of the packet, a receiver can calculate the index of each segment of the evidence. Since the evidence contains both a segment's hash value and the number of times to execute the hash, the receiver can individually authenticate each packet. Therefore, a receiver can verify reordered packets without delay.

4.1.5. Denial of Service Resistance

An adversary can initiate various types of denial of service (DoS) attack against its target, depending on which resource of its target it is attempting to deplete. In a network environment, the adversary can attack the sender, the network infrastructure, or the receiver. Because the sender must deliver the current temporal public key to its receivers, an adversary can launch a DoS attack against the sender by repeatedly requesting the TPK. Secure distribution of the TPK is beyond the scope of this work, thus, we do not examine this situation. Moreover, we also do not consider DoS attack which exhaust network bandwidth. However, we must investigate the possibility of an adversary launching a DoS attack on the receiver's computational and storage resources. Since our scheme only relies on fast one-way hash functions for its cryptographic primitive, a receiver can quickly verify a received packet's evidence. In addition, a receiver does not buffer forged packets since it can immediately check the packet's validity with the evidence. Thus, our scheme can resist DoS attacks that try to consume a receiver's computation and storage resources.

4.2. Overhead Comparison

In this section, we compare the computation, communication, and storage costs of our scheme to PARM.

4.2.1. Computation

To verify a packet, PARM must hash each segment of a packet's evidence the number of times specified in the usage table. Thus, validation consumes $1 + \sum_{i=0}^{p-1} U_i$ hash operations per packet. Because the receiver recalculates hash operations for each packet, this technique wastes a considerable number of computations. This trend is especially detrimental towards the end of a long hash chain. Under the

best-case scenario, our scheme simply requires one hash operation per packet segment. That is, verifying a packet uses $1+p$ hash operations. While operating in near optimal conditions in which packets arrive slightly out-of-order, our scheme requires significantly fewer computations than PARM, since it only hashes each segment of evidence a few times. A receiver need only hash each segment to the nearest hash value within the window of current hash values. For all remaining situations, our scheme expends the same amount of computation cost as PARM.

4.2.2. Communication

The communication overhead per packet, which consists of the evidence, remains constant in size for both PARM and our scheme. The overhead of PARM depends on the number of segments p per packet and the size of each TSK element n , that is, a cost of $p*n$ bits. Our scheme requires an additional $p*\lceil\lg(l)\rceil$ bits for the usage number of each segment and $\lceil\lg(q)\rceil$ bits for the sequence number. Thus, the total overhead per packet is $p*(n+\lceil\lg(l)\rceil)+\lceil\lg(q)\rceil$ bits. Since the additional overhead grows logarithmically, it is not a significant increase.

4.2.3. Storage

In both PARM and our scheme, the sender stores a set of temporal secret key chains. The sender generates k chains of length l with n -bits per TSK elements; thus, the set of TSK chains totals $k*l*n$ bits of storage. Moreover, the sender tracks the usage amount of each TSK chain; therefore, both schemes require $k*\lceil\lg(l)\rceil$ bits of additional space.

These two schemes differ in the size of the space used by the receiver. To store the TPK in PARM, the receiver uses a buffer size of $k*n$ bits. Because a receiver must also remember the number of times each TSK chain is used, it must set aside $k*\lceil\lg(l)\rceil$ bits of space. In contrast, a receiver implementing our scheme does not keep track of a usage table. On top of the storage needed for the TPK, however, the

receiver must buffer $w * k * n$ bits of space for the window of w current hash values per chain. To prevent replay attacks, it also must set aside $q * \lceil \lg(q) \rceil$ bits to remember the expended sequence numbers. Because PARM requires time synchronization between the sender and receiver, its receiver cannot validate packets until it obtains all previous packets, which can lead to buffer overload. Conversely, our scheme allows a receiver to immediately authenticate any packet it receives despite out-of-order or lost packets.



5. Conclusion

In this paper, we propose a multicast authentication scheme that effectively resists pollution attacks, a powerful class of denial of service attack. Due to its unsynchronized nature, our proposed scheme can readily tolerate packet loss or out-of-order packets by independently verifying received packets. Therefore, the receiver has no need to buffer packets while awaiting validation. Additionally, our scheme utilizes hash functions to maintain a lightweight computational overhead during evidence generation and verification. Finally, we suggest a key renewal plan that guarantees no reuse of hash values, yet curtails the waste of unused keys. By employing our scheme, other signature amortization schemes that depend on fault-tolerant algorithms can successfully resist pollution attacks and accept packet loss.

Based upon our requirements of a multicast authentications scheme, we provided a protocol analysis in terms of packet authenticity, packet loss robustness, and denial of service resistance. Furthermore, we evaluated our scheme against PARM, illustrating its improved computational performance while offering an enhanced feature set.

References

- [1] Y.-J. Lin, S. Shieh, and W. W. Lin, "Lightweight, Pollution-Attack Resistant Multicast Authentication Scheme," presented at Proceedings of the ACM Symposium on Information, Computer and Communications Security, Taipei, Taiwan, 2006.
- [2] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. D. Tygar, "Distillation codes and applications to DoS resistant multicast authentication," presented at Proceedings of the Network and Distributed System Security Conference (NDSS '04), San Diego, CA, 2004.
- [3] J. Pieprzyk, H. Wang, and C. Xing, "Multiple-time signature schemes against adaptive chosen message attacks," presented at 10th Annual International Workshop in Selected Areas in Cryptography (SAC '03), Ottawa, Canada, 2003.
- [4] J. M. Park, E. K. P. Chong, and H. J. Siegel, "Efficient multicast stream authentication using erasure codes," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, pp. 258-285, 2003.
- [5] A. Pannetrat and R. Molva, "Efficient multicast packet authentication," presented at Proceedings of the Network and Distributed System Security Conference (NDSS '03), San Diego, CA, 2003.
- [6] D. Song, D. Zuckerman, and J. D. Tygar, "Expander graphs for digital stream authentication and robust overlay networks," presented at Proceedings of the IEEE Symposium on Security and Privacy (S&P '02), Berkeley, CA, 2002.
- [7] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," presented at the 7th Australasian Conference on Information Security and Privacy (ACISP '02), Melbourne, Australia, 2002.

- [8] J. M. Park, E. K. P. Chong, and H. J. Siegel, "Efficient multicast packet authentication using signature amortization," presented at Proceedings of the IEEE Symposium on Security and Privacy (S&P '02), Berkeley, CA, 2002.
- [9] M. G. Luby, "LT codes," presented at Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '02), Vancouver, Canada, 2002.
- [10] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," presented at Proceedings of the Network and Distributed System Security Conference (NDSS '01), San Diego, CA, 2001.
- [11] A. Perrig, "The BiBa one-time signature and broadcast authentication protocol," presented at Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS '08), Philadelphia, PA, 2001.
- [12] S. Miner and J. Staddon, "Graph-based authentication of digital streams," presented at Proceedings of the IEEE Symposium on Security and Privacy (S&P '01), Berkeley, CA, 2001.
- [13] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Transactions on Information Theory*, vol. 47, pp. 569-584, 2001.
- [14] P. Golle and N. Modadugu, "Authenticating streamed data in the presence of random packet loss," presented at Proceedings of the Network and Distributed System Security Conference (NDSS '01), San Diego, CA, 2001.
- [15] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," presented at Proceedings of the IEEE Symposium on Security and Privacy (S&P '00), Berkeley, CA, 2000.
- [16] C. K. Wong and S. S. Lam, "Digital signatures for flows and multicasts," presented at Proceedings on the 6th International Conference on Network

Protocols (ICNP '98), Austin, TX, 1998.

- [17] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," presented at Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97), El Paso, TX, 1997.
- [18] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *Journal of the ACM (JACM)*, vol. 36, pp. 335-348, 1989.
- [19] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," presented at Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89), Seattle, WA, 1989.
- [20] R. C. Merkle, "Protocols for public key cryptosystems," presented at Proceedings of the IEEE Symposium on Security and Privacy (S&P '80), Berkeley, CA, 1980.
- [21] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, pp. 300-304, 1960.