TIM: A Trapdoor Hash Function-based Authentication Mechanism for Streaming Applications

Seog Chung Seo[1], and Taek-Young Youn[2]
[1] The Affiliated Institute of ETRI
Daejeon, Republic of Korea
[e-mail: seoseogchung82@gmail.com]
[2] ETRI
Daejeon, Republic of Korea
[e-mail: taekyoung@etri.re.kr]
*Corresponding author: Taek-Young Youn

Received September 21, 2017; revised December 5, 2017; accepted December 15, 2017; published June 30, 2018

Abstract

Achieving efficient authentication is a crucial issue for stream data commonly seen in content delivery, peer-to-peer, and multicast/broadcast networks. Stream authentication mechanisms need to be operated efficiently at both sender-side and receiver-side at the same time because of the properties of stream data such as *real-time* and *delay-sensitivity*. Until now, many stream authentication mechanisms have been proposed, but they are not efficient enough to be used in stream applications where the efficiency for sender and receiver sides are required simultaneously since most of them could achieve one of either sender-side and receiver-side efficiency. In this paper, we propose an efficient stream authentication mechanism, so called `TIM`, by integrating Trapdoor Hash Function and Merkle Hash Tree. Our construction can support efficient streaming data processing at both sender-side and receiver-side at the same time differently from previously proposed other schemes. Through theoretical and experimental analysis, we show that `TIM` can provide enhanced performance at both sender and receiver sides compared with existing mechanisms. Furthermore, `TIM` provides an important feature for streaming authentication, the resilience against transmission loss, since each data block can be verified with authentication information contained in itself.

Keywords: Stream authentication, Trapdoor Hash Function, Merkle Hash Tree, Digital Signature Algorithm (DSA), Signature amortization, online/offline signature.

1. Introduction

Nowadays many network applications or services involve distribution of contents like softwares, games, streaming data, digital audio, videos, and live news feeds through distributed networking technologies such as multicast networks, P2P (peer-to-peer) networks, and content delivery networks (CDNs). For secure delivery of these contents in the aforementioned networks, the authentication has to be offered as a primary requirement. Especially, for securing stream contents which are delay-sensitive or real-time, the authentication mechanism must be efficiently operated at both sender and receiver sides. In addition, it needs to be resilient against transmission loss, and the amount of additional authentication information should be minimal.

Until now, many researchers have proposed several stream authentication mechanisms aiming at reducing both computation and communication overhead related with securing individual blocks in a stream [4,5,6,7,8,9,19]. Among several mechanisms, recently, S. Chandrasekhar et al. [19] proposed an efficient stream authentication technique based on trapdoor hash function (THF), called DL-SA, for secure content delivery in CDNs. In fact, since DL-SA is a kind of online/offline signature scheme [11,12,13,14,15,16,17,18,19,20] using trapdoor hash function, the online signing can be efficiently processed by shifting computational burden to offline phase. They utilize these characteristics to design a signature amortization technique, which computes a signature on the trapdoor hash of the first block and amortizes it over remaining blocks by finding their trapdoor collisions with the trapdoor hash of the first signed block. The receiver verifies the authenticity of a block by computing its corresponding trapdoor hash value with a public hash key and comparing it with a previously proven trapdoor hash value. While DL-SA mechanism achieved small and constant memory/computation costs at the sender side, the verification cost at the receiver side is still expensive. It utilized the batch verification approach to alleviate the overhead of verification at the receiver side. Even if the batch verification method can reduce the computation cost, it requires a receiver to buffer data blocks before authenticating them, which attenuates the usefulness of DL-SA in real-time or delay-sensitive applications. Furthermore, this limits the usefulness of DL-SA on battery-powered smart devices or resource-constrained IoT devices.

In this paper, we propose an efficient stream authentication mechanism, so called TIM[1], which can significantly reduce the cost for signing and verification cost at the same time. The proposed mechanism makes use of Trapdoor Hash Function (THF) and Merkle Hash Tree (MHT) [1,2] and it integrates them for optimized signing and verification. The signing process in TIM is divided into offline phase and online phase. Signatures of base trapdoor hash values are computed at offline phase before transmitting stream data. When transmitting data blocks in a stream data, TIM generates authentication information for each data block with negligible cost. With respect to data block verification, for verifying k data blocks belonging to same MHT block, only the first block is verified with trapdoor hash verification and subsequent $(k-1)$ blocks are verified by comparing their hash values with the cached hash values authenticated in the previous trapdoor hash verification, while DL-SA needs at least one trapdoor hash verification per data block. Finally, since TIM does not require any receiver side buffering, it is a more appropriate candidate as an authentication method for secure delivery of

[1] **T**rapdoor hash function-based stream authentication mechanism **I**ntegrated with **M**erkle hash tree

stream data considering recently commonly used battery-powered smart devices. As far as we know, this is the first work seamlessly integrating the concept of MHT to trapdoor hash function-based authentication.

The rest of this paper is organized as follows. We briefly review some related techniques and previously proposed works in Section 2. Then, we present the proposed stream authentication method, TIM, in Section 3. The security and theoretical performance are analyzed in Section 4. We present the performance of real implementation of TIM and compare it with DL-SA in Section. Section 6 concludes this paper.

2. Related Work

In this section, we describe two essential techniques, Merkle Hash Tree and trapdoor hash functions for explaining our work, and related works on stream authentication.

2.1 MHT-based Signature Amortization

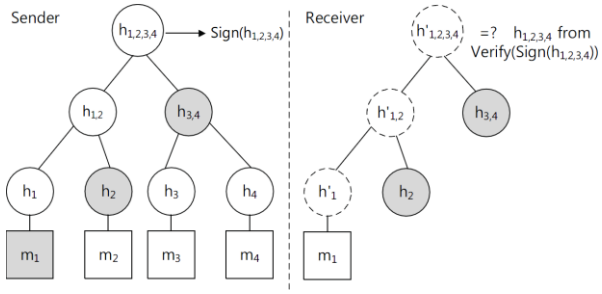

Fig. 1. Signing and verification process using MHT with four data blocks, $k = 4$ (Gray circles refer to the hash values to be transmitted or received and dotted circles mean that the hash values are not received but computed with witness information.

To reduce the overhead of signature signing and verification process, the Merkle Hash Tree (MHT) [1,2,4] has been widely used. A sender buffers k data blocks and signs only the root hash node of the binary hash tree about them. The sender sends a packet including the data block, its *witness* (the hash tree path to the root node), and the root signature to receivers. When the receiver receives the packet, he/she can verify its validity by computing the root hash value of the hash tree using the received data block and its witness, and comparing the computed root hash value with the root hash value from the root signature verification. **Fig. 1** shows the signing and verification process using MHT for authenticating four data blocks. The sender constructs a binary hash tree (so called Merkle Hash Tree) dealing with the four data blocks and signs the root hash node ($Sign(h_{1,2,3,4})$). About the m_1 data block, the sender transmits (m_1, h_2, $h_{3,4}$, $Sign(h_{1,2,3,4})$) (h_2 and $h_{3,4}$ form the *witness* field in the packet). When the receiver gets the packet (m_1, h_2, $h_{3,4}$, $Sign(h_{1,2,3,4})$), he/she verifies the validity of the packet through the signature verification. In the progress of signature verification, the root hash value ($h'_{1,2,3,4}$) is computed from the received m_1, h_2, and $h_{3,4}$, and then compared with the original root hash value ($h_{1,2,3,4}$) embedded in the signature $Sign(h_{1,2,3,4})$. Thus, an alteration to any fields in the packet leads to the failure of signature verification. If the signature verification is proven to be valid, the validity of both the data block and the hash values including h_2, $h_{1,2}$, $h_{3,4}$ and

$h_{1,2,3,4}$ is proven. The receiver caches the verified hash values at the authenticated hash table (AHT) for fast verification of the other data blocks that belong to the same hash tree.

2.2 Trapdoor Hash Functions

Trapdoor hash functions (TH) introduced by Krawczyk and [2] is related with a public hash key HK and a private trapdoor key TK . It has a unique property of being easy to generate collisions between hashes of different messages with TK , while being difficult when only HK is known [13]. For example, with only HK , it is hard to find two messages m , m' and auxiliary numbers r , r' satisfying $TH(m,r)=TH(m',r')$, nevertheless, with HK and TK , it is easy to find a collision value r satisfying $TH(m,r)=TH(m',r')$ given m , m' and r' .

Shamir and Tauman [13] developed *hash-sign-switch* paradigm that can convert any signature scheme into an online/offline signature scheme [11]. The main concept of online/offline schemes is shifting computational burden from online phase to offline phase when generating signatures. Thus, the computational overhead at online phase is much lighter than that at offline phase (At offline phase, typical signature schemes can be utilized). For instance, in the work of [13], at offline phase, a signature can be computed for $TH(m',r')$ of a random message m' and a random number r' . Given an actual message m to be signed, the aim of online phase is to find a collision value r that satisfies $TH(m,r)=TH(m',r')$ by using trapdoor key TK (Without the knowledge of TK , finding r is computationally infeasible). Thus, the signature of m is composed of r and $TH(m',r')$. The cost for computing r at online phase is much lighter than that for computing a signature at offline phase. From the work of [12] and [13], several researches have been conducted with respect to developing various online/offline signature schemes and their security analysis [14, 15, 16, 17, 18, 19, 20, 23, 24, 25]. In 2013, Yi Sun et al. extended the work of [19] and presented elliptic curve discrete logarithm based trapdoor hash function [23]. In 2014, S. Chandrasekhar et al. proposed multi-trapdoor hash function that allows multiple entities to compute a trapdoor hash collision with a given hash value [20]. Recently, they proposed multi-trapdoor hash function-based signcryption [24] and applied multi-trapdoor hash function for efficient query authentication on cloud-based storage system where data is generated by multiple sources and retrieved by the clients [25].

2.3 Stream Authentication Methods

Until now, many researches have been conducted for designing efficient authentication mechanisms for streaming applications including multicast and broadcast. Their main concern is to reduce both computation and transmission overhead in authenticating individual blocks constructing a stream while providing a certain level of resilience to data block losses.

Although signing and verifying each block in a stream individually provides robustness against data losses, it imposes heavy amount of computation for authenticating data stream due to algorithmic property of asymmetric-key algorithms. In order to reduce computational overhead, several signature-based authentication mechanisms have been proposed such as `WL` [4], `EMSS` [5], `AC` [6], `SAIDA` [7], `MABS` [9] and `DL-SA` [19]. Their main idea is to amortize a signature over multiple blocks in a stream.

`EMSS` and `AC` apply hash chains to authenticating data blocks in a stream. Even though hash chain-based authentication is very efficient, it is vulnerable to transmission loss and out-of-order transmission. Thus, `EMSS` and `AC` place multiple hashes in each block to mitigate transmission loss and our-of-order transmission. Rather than using redundant multiple hashes,

SAIDA make use of erasure codes considering space optimality. Thus, even though some data blocks are lost during transmission, the information for authenticating data blocks can be recovered. Since EMSS, AC, and SAIDA are probabilistic authentication schemes, each block cannot be verified independently. In other words, the verification success rate depends on the transmission loss probability and the amount of buffered information at the receiver side.

WL is based MHT. In WL, a hash tree is built for multiple data blocks and the only root hash node is signed. For a data block to be individually verifiable in WL, each data block needs to carry its own authentication information such as the generated signature and *witness*. To speed up the signing and verifying operations, WL integrates Feige-Fiat-Shamir digital signature scheme to MHT when signing the root hash node and verifying the signature. Since WL is a deterministic scheme, each data block in WL can be verified independently at the expense of transmission overhead for additional information and sender side buffering.

MABS is a deterministic multicast authentication mechanism and it signs and verifies each data block in a stream individually to prevent DoS attack. Since MABS generates signature for each data block in a stream and verifies them individually, it suffers from signature verification overhead. To mitigate signature verification overhead, MABS proposes a batch signature verification method. Even though it could reduce signature verification overhead about in half, it still requires exponentiation computation for each data block.

Recently, S. Chandrasekhar et al. [19] proposed an efficient stream authentication technique based on trapdoor hash function, called DL-SA, for stream authentication aiming at both reducing signing delay and minimizing cost for signing at the sender, while providing robustness against transmission loss. DL-SA computes a signature on the trapdoor hash of the first block and amortizes it over remaining blocks by finding their trapdoor hash collisions (computing r value in Section 2.2) with the first signed block's trapdoor hash value. Since compared with typical signature generation, finding collision is very fast, DL-SA achieves efficient and constant cost for signing. However, it suffers from verification overhead at the receiver side. In DL-SA, verifying a block requires at least two modular exponentiations, while finding collision calls for only a few modular multiplications. To alleviate verification overhead at the receiver side, DL-SA applies a batch verification mechanism. Even if the batch method can reduce computation cost for verification, it requires buffering at the receiver-side and its cost is still large for battery-powered smart devices. Furthermore, DL-SA with the batch verification method has difficulty in identifying which data blocks contribute to the failure when the batch verification for authenticating several data blocks fails.

In 2007, Deng and Yang presented and end-to-end authentication mechanism from multimedia server to end users through intermediary proxies [22]. Even though their scheme used trapdoor hash function and Merkle Hash Tree, however, it utilizes each of them separately in the server and proxy. Thus, Deng and Yang's mechanism did not achieve seamless integration of trapdoor hash function and Merkle Hash Tree for both efficient signing and efficient verification. Actually, the multimedia server generates a RSA-based signature of the multimedia data. At this time, it utilized Merkle Hash Tree for efficient signing. The proxy removes some data components from the original set for content downscaling and generates trapdoor hash collision that can authenticate the downscaled multimedia data. Furthermore, the cost for trapdoor hash collision is much larger than that in our mechanism. Thus, Deng and Yang's mechanism is proper for authenticating constant multimedia data like images rather than streaming application where real-time and delay-sensitivity are important.

3. Proposed Authentication Mechanism

The goal of TIM is to minimize signing at sender-side and verification cost at receiver-side at the same time. Since it uses the root hash of MHT when computing trapdoor hash collisions, it requires the sender to buffer k data blocks before signing them[2]. It is mainly based on *DL-Schnorr* signature scheme [3], but the concept of the proposed mechanism can be applied to any trapdoor hash function-based authentication mechanisms.

3.1 Preliminaries

Table 1 describes major notations used in this paper. All entities agree upon common system public parameters $params = \langle p, q, \alpha, \mathrm{H}, \mathrm{G} \rangle$. We assume that the stream $\mathcal{S}$ is continuously generated in the form of segments like (s_0, s_1, …, s_v, …), and each segment is composed of n data blocks m_0, m_1, …, m_{n-1}. **Fig. 2** depicts the structure of the streaming data in TIM.

Table 1. Notations.

Symbols	Definitions
S	Streaming data which is continuously generated in the form of segments
n	the number of data blocks in a segment
k	Merkle Hash Tree(MHT) size
l	the number of MHT blocks in a segment consisting n data blocks, $n = k \cdot l$
$params$	public system parameters including p, q, and α where p and q are 1,024-bit and 160-bit primes, respectively, $q \mid p-1$ and α is an element of order q in Z_p^*
H, G	$\{0,1\}^* \mapsto Z_q^*$ are cryptographic hash functions
x, X	sender's long-term (private, public) key pair where $x \in_R \mathbb{Z}_q^*$ and $X = \alpha^x \in \mathbb{Z}_p^*$
k_v, r_v	sender's ephemeral (private, public) key pair where $k_v \in_R \mathbb{Z}_q^*$, $r_v = \alpha^{k_v} \in \mathbb{Z}_p^*$, and v is the segment index
$\bar{y}$, $\bar{Y}$	sender's long-term (trapdoor, hash) key pair where $\bar{y} \in_R \mathbb{Z}_q^*$ and $\bar{Y} = \alpha^{\bar{y}} \in \mathbb{Z}_p^*$
y_i, Y_i	sender's ephemeral (trapdoor, hash) key pair such that $y_i \in_R \mathbb{Z}_q^*$ and $Y_i = \alpha^{y_i} \in \mathbb{Z}_p^*$ where $0 \le i \le l-1$
$h_{A,B}$	computed hash value of the concatenation of A and B such as H(A||B)
$RH_{ik,(i+1)k-1}$	a root hash value of i-th MHT of size k in a segment. Its leaf nodes consists of hash values of m_{ik}, $m_{ik+1}, \ldots, m_{(i+1)k-1}$ data blocks
$TH_Y(m,z)$	trapdoor hash value of message m and z such that $TH_Y(m,z) = ((\alpha^{m+yz} \bmod p) \bmod q)$ where y and Y are trapdoor key and hash key, respectively

Even though each data block has its segment index to which it belongs, for the simplicity we omit the segment index in the figure. For example, the v-th segment S_v consists of $m_{v,0}, m_{v,1}, \ldots, m_{v,n-1}$. We also assume that the number of data blocks is multiple of MHT size k ($n = l \cdot k$). Namely, there are l MHT blocks in a segment.

3.2 Proposed Mechanism Specification

TIM consists of offline process and online process. Offline process includes system setup and offline signing. Online process is composed of online data block signing at the sender side and

[2] Even if typically 2's power value is used for k, numbers which are not 2's power can be applied by using the method from [21].

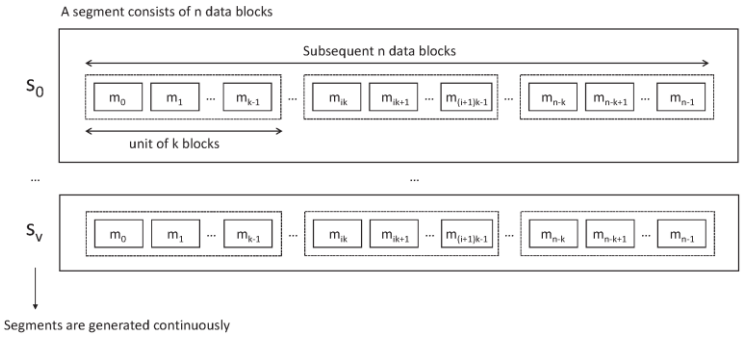

Fig. 2. Streaming data structure in TIM.

data block verification at the receiver side. For simplicity of description, we assume that the stream $\mathcal{S}$ is partitioned into w segments, each segment is composed of n blocks, and n is multiple of MHT size k even though w, n, and k values need to be determined considering the performance of system and network.

3.2.1 System Setup Specification

Algorithm 1. System setup function Sys_Setup()

Require: Common system public parameters $params = \langle p, q, \alpha, \mathrm{H}, \mathrm{G} \rangle$.

Ensure: Key materials, the number of blocks n in a segment and MHT size k.

1. Generate long-term (private, public) key pair as (x, X): $x \in_R \mathbb{Z}_q^*$ and $X = \alpha^x \in \mathbb{Z}_p^*$.
2. Generate long-term (trapdoor, hash) key pair as $(\overline{TK}, \overline{HK}) = (\bar{y}, \bar{Y})$: $\bar{y} \in_R \mathbb{Z}_q^*$ and $\bar{Y} = \alpha^{\bar{y}} \in \mathbb{Z}_p^*$.
3. Choose n to determine the number of blocks in a segment.
4. Determine MHT size k such that k is power of 2 and $n = k \cdot l$.
5. Generate ephemeral (private, public) key pairs as (k_v, r_v):
   $k_v \in_R \mathbb{Z}_q^*$ and $r_v = \alpha^{k_v} \in \mathbb{Z}_p^*$, where $0 \le i \le w-1$.
6. Generate ephemeral (trapdoor, hash) key pairs as $(TK_i, HK_i) = (y_i, Y_i)$:
   $y_i \in_R \mathbb{Z}_q^*$ and $Y_i = \alpha^{y_i} \in \mathbb{Z}_p^*$, where $0 \le i \le l-1$.

Return: n, k, (x, X), $(\bar{y}, \bar{Y})$, (y_i, Y_i) where $0 \le i \le l-1$, and (k_v, r_v) where $0 \le v \le w-1$.

To provide the key exposure resistance, S. Chandrasekhar et al. utilized different trapdoor/hash key pairs for each data block in a segment [19]. Thus, their scheme requires at least n trapdoor/hash key pairs. However, TIM requires only l different trapdoor/hash key pairs since it applies MHT for n data blocks and computes l collision values making their trapdoor hash values colliding to the segment's base trapdoor hash value. Thus, TIM chooses ephemeral trapdoor key $y_i \in_R \mathbb{Z}_q^*$ and computes the corresponding ephemeral hash key $Y_i = \alpha^{y_i} \in \mathbb{Z}_p^*$, and stores the pair ($y_i$, Y_i) where $0 \le i \le l-1$. In other words, (y_i, Y_i), a pair of trapdoor key and hash key, is used to compute a collision value which incurs i-th MHT block's trapdoor hash value to collide to the segment's base trapdoor hash value. Now that these trapdoor keys and hash keys will be reused when processing each segment, this process is just a one-time operation. **Alg. 1** generates key materials including long-term (private, public) key pair, long-term (trapdoor, hash) key pair, w pairs of ephemeral (private, public) key pairs and l ephemeral (trapdoor, hash) key pairs which are necessary for offline/online

signing and verification, and determines system parameters such as segment size n, MHT size k, which can be conducted at offline. We assume that the public parameters *params* for DSA have been already generated and shared among network participants. Note that l pairs of ephemeral (trapdoor, hash) key are reused for signing each segment.

3.2.2 Offline Signing Specification

The goal of offline signing process is generating a root of trust for each segment. In other words, it generates a base trapdoor hash value for each segment with long-term (trapdoor, hash) key pair and signs it with long-term private key and ephemeral (private, public) key pair. By verifying the signature of a segment, a receiver can trust the verified trapdoor hash value and authenticate data blocks in the segment by comparing their trapdoor hash values to the base trapdoor hash value. Since random data blocks are used to compute segments' base trapdoor hash values, offline signing process can be conducted in offline phase before starting online signing phase which signs real data blocks belonging to stream data.

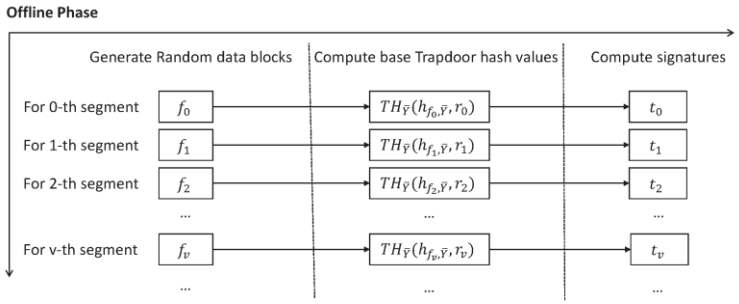

Fig. 3. Offline signing phase in TIM.

Algorithm 2. TIM Offline Signing process Offline_Signing()

Require: The number of segments w in a stream data $\mathcal{S}$, long-term private key x, long-term (trapdoor, hash) key pair=($\bar{y}$, $\bar{Y}$), and ephemeral (private, public) key pair (k_v, r_v) where v from 0 to $w-1$.

Ensure: List of signatures for authenticating each segment's base trapdoor hash value.

1. **for** v from 0 to $w-1$ **do**
2. Generate random data block f_v.
3. **end for**
4. /* Generate base trapdoor hash value of each segment and sign it */
5. **for** v from 0 to $w-1$ **do**
6. Compute v-th segment's base trapdoor hash value:
   $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v) \leftarrow (\alpha^{\exp} \bmod p) \bmod q$, where $\exp = h_{f_v,\bar{Y}} + \bar{y}r_v$, $h_{f_v,\bar{Y}} = \mathrm{H}(f_v \| \bar{Y})$.
7. Generate signature of the base trapdoor hash value with DL-Schnorr [3]:
   $t_v \leftarrow k_v + x\mathrm{G}(TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v) \| f_v \| r_v) \bmod q$.
8. **end for**

Return: List of signature containing t_v where $0 \le v \le w-1$.

Fig. 3 shows an example of offline signing process in TIM. Firstly, offline signing phase generates random data blocks ($f_0, f_1, \ldots, f_v, \ldots$) which are used to compute base trapdoor hash values. Namely, f_v is used to compute the base trapdoor hash value in v-th segment. Then, TIM computes the base trapdoor hash value of each segment. When computing v-th base

trapdoor hash value $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$, hash value $h_{f_v,\bar{Y}}$, ephemeral public key r_v, and long-term private key $\bar{y}$ are used. Next, the computed base trapdoor hash values are signed with DL-Schnorr signature scheme. t_v is the signature of the v-th trapdoor hash value.

Alg. 2 shows the detail of offline signing process in TIM. Offline signing requires long-term private key x, long-term (trapdoor, hash) key pair. Step 6 computes a base trapdoor hash value $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$ for v-th segment and it requires a modular exponentiation. This step computes a trapdoor hash value of $h_{f_v,\bar{Y}}$ rather than original f_v data block in order to make a relation of data block f_v and long-term hash key $\bar{Y}$. Step 7 signs the generated base trapdoor hash value with sender's long-term private key x and ephemeral private key k_v. **Alg. 2** returns the list of signatures which can authenticate each segment's base trapdoor hash values.

3.2.3 Online Signing Specification

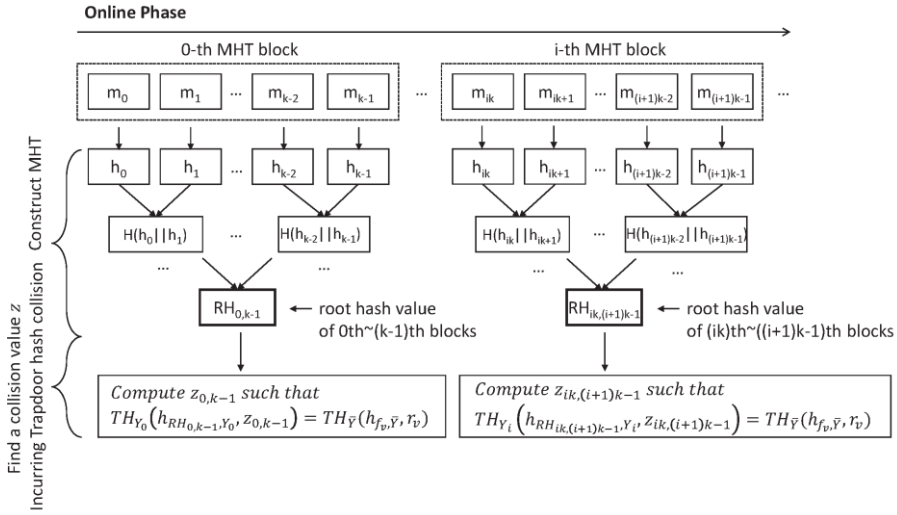

Fig. 4. Online signing phase in TIM.

Online signing process in TIM generates authentication information that makes each data block in a stream data verified independently. Online signing process makes use of base trapdoor hash values previously computed and stored in offline signing process when computing collision values. TIM applies collision value computation to the root hash value of MHT consisting of k data blocks rather than computing each data block's collision value.

Fig. 4 describes an example of the online signing process for v-th segment in TIM. For k data blocks $m_{ik},\ldots,m_{(i+1)k-1}$, at first MHT is constructed and then, a collision value $z_{ik,(i+1)k-1}$ is computed. For example, the root hash of k data blocks $m_{ik},\ldots,m_{(i+1)k-1}$ is $RH_{ik,(i+1)k-1}$. Then, TIM computes a collision value $z_{ik,(i+1)k-1}$ incurring a collision for the trapdoor hash function such that $TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$ where $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$ is the base trapdoor hash value in v-th segment and $TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1})$ is the trapdoor hash value of $h_{RH_{ik,(i+1)k-1},Y_i}$ which is the hash value of the concatenation of root hash value $RH_{ik,(i+1)k-1}$ and ephemeral hash key Y_i. Note that only the sender who has a pair of ephemeral trapdoor y_i and

ephemeral hash keys Y_i can compute the collision value with a few multiplications over $\mathbb{Z}_q^*$.

Algorithm 3. TIM Online Signing process Online_Signing()

Require: Stream data $\mathcal{S}$ consisting of w segments, key materials including long-term (trapdoor, hash) key pair ($\bar{y}$, $\bar{Y}$), ephemeral public keys r_v where v from 0 to $w-1$, ephemeral (trapdoor, hash) key pairs (y_i, Y_i) where i from 0 to $l-1$.

Ensure: Send data blocks in each segment of $\mathcal{S}$ and their corresponding authentication information.

1. **for** v from 0 to $w-1$ **do**
2. Get t_v and $h_{f_v,\bar{Y}}$ which was generated at Offline Signing process from system signature pool.
3. Send v-th segment's signature $\langle f_v, t_v, r_v \rangle$ to receivers.
4. **for** i from 0 to $l-1$ **do**
5. Construct MHT of k data blocks $m_{ik},\ldots,m_{(i+1)k-1}$ and store root hash value $RH_{ik,(i+1)k-1}$ and hash values in MHT.
6. Compute a collision value $z_{ik,(i+1)k-1}$ satisfying

$$TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v) : z_{ik,(i+1)k-1} \leftarrow y_i^{-1}(h_{f_v,\bar{Y}} - h_{RH_{ik,(i+1)k-1},Y_i} + \bar{y}r_v) \bmod q$$

where $h_{RH_{ik,(i+1)k-1},Y_i} = \mathrm{H}(RH_{ik,(i+1)k-1} \| Y_i)$.

7. **for** j from ik to $(i+1)k-1$ **do**
8. Construct authentication information $AI_j \leftarrow \langle z_{ik,(i+1)k-1}, wit_j, Y_i \rangle$.
9. Send $\langle m_j, AI_j \rangle$ to receivers.
10. **end for**
11. **end for**
12. **end for**

Since we assume that the number of data blocks in a segment is a multiple of MHT size k ($n = k \cdot l$), there are l MHT blocks in a segment. Thus, the process of constructing MHT and computing collision value are repeated l times for each segment. **Alg. 3** is general online signing process in TIM. Even though the algorithm assumes that a stream data consists of w segments, it can be operated on continuously generated segments. Step 2 gets t_v which is the signature of v-th segment's base trapdoor hash value and $h_{f_v,\bar{Y}}$ which will be used in collision value computation from the system pool. Then, it first sends $\langle f_v, t_v, r_v \rangle$ containing the data block f_v, the signature of base trapdoor hash value t_v, and ephemeral public key r_v to receivers. The signature t_v is the root of trust for authenticating data blocks in v-th segment. If t_v passes signature verification process, the receivers can accept a data block as authentic one if its trapdoor hash value is identical to the base trapdoor hash value. Step 5–6 is the collision value generation process. As aforementioned, TIM applies collision value computation to the representative root hash value of MHT composed of k data blocks instead of computing each data block's collision value for signing and verification efficiency. Thus, Step 5 constructs MHT of k data blocks, and stores root hash value $RH_{ik,(i+1)k-1}$ and hash values from the MHT construction. Then, Step 6 computes a collision value $z_{ik,(i+1)k-1}$ satisfying the relation of $TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$. For computing collision value $z_{ik,(i+1)k-1}$, Step 6 makes the root hash value related to ephemeral hash key Y_i by hashing their concatenation. Then, Step 6 computes a collision value $z_{ik,(i+1)k-1}$ of $h_{RH_{ik,(i+1)k-1},Y_i}$ such that its trapdoor hash

value collides to v-th segment's base trapdoor hash value. Since the aim of the collision value is making the trapdoor hash value of $h_{RH_{ik,(i+1)k-1},Y_i}$ is identical to the base trapdoor hash value $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$, its computation requires $h_{f_v,\bar{Y}}$. Furthermore, since collision value computation requires long-term trapdoor key $\bar{y}$, ephemeral trapdoor key y_i, only the sender can compute the collision as long as private information is not revealed to adversaries. Even though naive implementation of collision value computation requires the modular inverse of y_i and the modular multiplication of $\bar{y}$ and r_v, they can be precomputed at the offline phase and reused at the online phase. Thus, computing collision value in Step 6 requires only one modular multiplication of y_i^{-1} and $(h_{f_v,\bar{Y}} - h_{RH_{ik,(i+1)k-1},Y_i} + \bar{y}r_v)$ by reusing precomputed values y_i^{-1} and $\bar{y}r_v$. After computing a collision value, Step 7–9 constructs each authentication information of k data blocks as the combination of a collision value $z_{ik,(i+1)k-1}$, block's witness information wit_j which composed of sibling hash values in the MHT, and ephemeral hash key Y_i, and sends each data block with the generated authentication information to receivers. Note that $z_{ik,(i+1)k-1}$ is the common collision value for k data blocks since it is computed on the root hash value of k data blocks' MHT.

3.2.4 Verification Specification

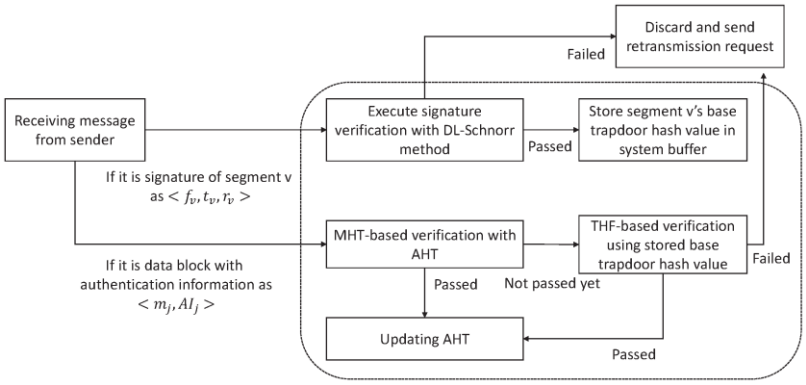

Fig. 5. Verification process of TIM at receivers.

TIM makes use of the properties of both MHT and THF for verifying data blocks. The property of MHT in TIM makes a data block verification efficient even though it requires previously authenticated hash values in AHT (Authenticated Hash Table). If the root hash value of the MHT is proven to be authentic, hash values used to compute it can be utilized to verify the validity of each data block belonging to the MHT. The property of THF in TIM enables receivers verify any data blocks individually in a segment by comparing its trapdoor hash value with the segment's base trapdoor hash value if the base trapdoor hash value is proven to be authentic. Since the overhead for computing trapdoor hash value is large, TIM applies THF-based verification, requiring the computation of trapdoor hash value, to only one of k data blocks in the same MHT and executes MHT-based verification to remaining $k-1$ data blocks.

Fig. 5 describes the verification process of TIM at receiver side. The verification process starts from receiving messages from a sender. There are two kinds of message: a data block

Algorithm 4. Signature Verification Process *Verify_Sig*() .

Require: data block f_v with its signature and related ephemeral public key as $\langle f_v, t_v, r_v \rangle$, long-term hash key $\overline{Y}$, and sender's public key X .

Ensure: Verification result (*Pass* or *Fail*).

1. Compute $h_{f_v,\overline{Y}} = \mathrm{H}(f_v \| \overline{Y})$.
2. Compute $h_{f_v,\overline{Y}}$'s trapdoor hash value: $TH_{\overline{Y}}(h_{f_v,\overline{Y}}, r_v) = (\alpha^{h_{f_v,\overline{Y}}} \overline{Y}^{r_v} \mod p) \bmod q$.
3. Execute DL-Schnorr signature verification:
   $r' = (\alpha^t X^{-h} \mod p) \bmod q$ where $h = \mathrm{G}(TH_{\overline{Y}}(h_{f_v,\overline{Y}}, r_v) \| f_v \| r_v)$.
4. **if** $r' \neq r_v \mod q$ **then**
5. *ret* = *Fail*
6. **else**
7. Save $TH_{\overline{Y}}(h_{f_v,\overline{Y}}, r_v)$ in the system buffer as v -th segment's base trapdoor hash value.
8. *ret* = *Pass* .
9. **end if**

Return: *ret*

with signature as $\langle f_v, t_v, r_v \rangle$ or a data block with authentication information as $\langle m_j, AI_j \rangle$. Since t_v is the signature of v -th segment's base trapdoor hash value, first of all, it needs to be verified before verifying data blocks in v -th segment. Actually, online signing process in **Alg. 3** firstly sends the signature in a segment and then sends data blocks in the segment. $\langle f_v, t_v, r_v \rangle$ is verified through DL-schnorr signature verification process. **Alg. 4** depicts the signature verification process. **Alg. 4** computes the trapdoor hash value of $h_{f_v,\overline{Y}}$ and then verifies the signature t_v . Both computing trapdoor hash value and verifying signature require the computation of modular exponentiations such as $((\alpha^{h_{f_v,\overline{Y}}} \overline{Y}^{r_v} \mod p) \bmod q)$ and $((\alpha^t X^{-h} \mod p) \bmod q)$. In TIM, these modular exponentiations can be sped up by using fixed-base exponentiation algorithm such as LL exponentiation method [10]. If the verification succeeds, the verified base trapdoor hash value is stored in system buffer since it is used to check the validity of MHT block's trapdoor hash value. When a data block with authentication information as $\langle m_j, AI_j \rangle$ is received, firstly TIM attempts MHT-based data block verification by using AHT. At first, it applies MHT-based verification by using AHT which contains previously authenticated hash values. Unless there are related hash values in the AHT, then it applies THF-based verification. MHT-based method requires hash values from the results of the other blocks verification, but it can be executed very fast with a few hash value comparisons. While THF-based method does not require any information contained in other blocks, it needs more complex computation than MHT-based method. Thus, if the $\langle m_j, AI_j \rangle$ is the first block in a MHT, there is no hash values which can be used for verifying m_j in AHT. Thus, TIM executes THF-based data block verification for the first block in the MHT. Since collision value is generated on the root hash value of a MHT block at the sender side, the trapdoor hash value is common for all data blocks belonging to the same MHT block. In other words, since the trapdoor hash values computed by using received data block, its witness, related hash key, and collision value are identical, the trapdoor hash value needs to be verified only once. In THF-based verification, the trapdoor hash value of the MHT's root hash value is

computed and it is compared to the base trapdoor hash value which is previously verified in signature verification process. If the THF-based verification succeeds, the hash values from computing the root hash value of the MHT are stored in AHT in order to utilize them for verifying other remaining data blocks belonging to the MHT. After the first data block in a MHT is verified with THF-based verification method, TIM is able to execute MHT-based verification method for remaining $k-1$ data blocks in the MHT since the AHT contains the MHT's hash values which have been verified.

Algorithm 5. Data block verification function *Verify_block*() .

Require: Segment index v in stream data where $0 \le v \le w-1$, data block with authentication information as $\langle m_j, AI_j = (z_{ik,(i+1)k-1}, wit_j, Y_i) \rangle$ where $ik \le j \le (i+1)k-1$ and $0 \le i \le l-1$.

Ensure: Verification result (*Pass* or *Fail*).

1. $ret \leftarrow Fail$.
2. /* Execute *MHT-based verification process**/
3. Compute $h_j = \mathrm{H}(m_j)$ and search h_j from the AHT.
4. **if** h_j found in AHT **then**
5. $ret \leftarrow Pass$.
6. **else**
7. **while** Until root hash $RH_{ik,(i+1)k-1}$ is computed **do**
8. Compute recursively upper hash values with h_j and wit_j .
9. **if** Computed hash value found in AHT **then**
10. $ret \leftarrow Pass$.
11. Update AHT with the computed hash values.
12. break loop.
13. **end if**
14. **end while**
15. **end if**
16. /*Execute *THF-based verification process**/
17. **if** $ret = Fail$ **then**
18. Compute $RH_{ik,(i+1)k-1}$, root hash of i -th MHT in v -th segment, with m_j and wit_j .
19. Compute $h_{RH_{ik,(i+1)k-1},Y_i} = \mathrm{H}(RH_{ik,(i+1)k-1} \| Y_i)$.
20. Compute $h_{RH_{ik,(i+1)k-1},Y_i}$'s trapdoor hash value using collision value $z_{ik,(i+1)k-1}$:

    $TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1}) = (\alpha^{\exp_1} Y_i^{\exp_2} \mod p) \bmod q$

    where $\exp_1 = h_{RH_{ik,(i+1)k-1},Y_i}$ and $\exp_2 = z_{ik,(i+1)k-1}$
21. Get v -th segment's base trapdoor hash value $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$ from system buffer.
22. **if** $TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v) = TH_{Y_i}(h_{RH_{ik,(i+1)k-1},Y_i}, z_{ik,(i+1)k-1})$ **then**
23. $ret = Pass$.
24. Update AHT with the computed hash values in MHT-based method.
25. **else**
26. $ret = Fail$
27. **end if**
28. **end if**

Return: *ret*

Alg. 5 describes the detail of data block verification. MHT-based verification is executed in Step 2–15 and firstly it computes data block m_j 's hash value h_j and searches whether there is same hash value in the AHT. If the same hash value is found in the AHT, this means that the

hash value of m_j has been previously verified and stored. Thus, weak authentication to m_j is proven. Unless there is no matched hash value in the MHT, Step 7–14 tries to find the related hash values in the AHT until the root hash value is computed. These steps compute the root hash value of the MHT with m_j's hash value and m_j's witness information wit_j and searches computed hash values of the MHT in AHT. If m_j succeeds to pass the MHT-based verification, TIM updates the AHT by reflecting the hash values which are computed during computing the root hash value. Otherwise, TIM executes THF-based verification. This situation occurs when m_j is the firstly received data block among data blocks of the MHT to which m_j belongs. The goal of THF-based verification is to check whether the trapdoor hash value of the MHT's root hash value is identical to the base trapdoor hash value. Thus, it first computes the root hash value $RH_{ik,(i+1)k-1}$ of the MHT consisting of $m_{ik},\ldots,m_{(i+1)k-1}$ in Step 18 and then computes the trapdoor hash value of $h_{RH_{ik,(i+1)k-1},Y_i}$ with the received collision value $z_{ik,(i+1)k-1}$. As aforementioned, the collision value has been computed on the root hash value of a MHT at the sender side, the computed hash values of any data blocks in the MHT are identical. Similar to the signature verification in **Alg. 3**, since computing a trapdoor hash value requires modular exponentiations, TIM applies LL exponentiation method [10] when computing $((\alpha^{h_{RH_{ik,(i+1)k-1},Y_i}} Y_i^{z_{ik,(i+1)k-1}} \bmod p) \bmod q)$ for fast processing. If the computed trapdoor hash value is identical to the base trapdoor hash value, then the THF-based verification succeeds and the AHT is updated by including computed hash values from computing the root hash value in Step 18. If both MHT-based verification and THF-based verification fail, then the receiver discards the data block and sends a retransmission request.

4. Analysis of Security and Computational Overhead

4.1 Security Analysis

Recall that we design a new streaming data authentication mechanism by combining two existing techniques, the amortized signature scheme in DL-SA and MHT. The security of amortized signature scheme is proven in [19] and the security is MHT is well-known, and thus we can have the following propositions.

Proposition 1. The security of amortized signature scheme is guaranteed by the security of the unforgeability of DL-Shnorr signature scheme, the collision resistance, and key exposure freeness of the underlying trapdoor hash function.

Proposition 2. The MHT is intractable if the underlying hash function is collision resistance.

Simply combining two secure schemes does not guarantee the security of the resulting scheme. Hence, to guarantee the security of our streaming data authentication mechanism, we prove that if our scheme is not secure then at least one of two underlying schemes is not secure. Since two components are proven to be secure, we can guarantee the security of our construction if our assertion is true. For the security analysis, we do not use oracle-based security proof since the approach is not suitable to analyze complex protocols such as our technique. In the literature, various approaches, such as hybrid arguments based security proof [26] and event based (a.k.a., classification based) security proof [19], have been introduced for provable

security of complex protocols. Among well-known approaches, for the provable security of our scheme, we will use event-based proof technique due to the simplicity of the approach.

Our basic idea for security proof is as follows. At first, we define an event which covers all successful attack scenarios that can be performed by any adversary. Then we will divide the event into three disjoint sub-events whose union can cover the full event, and then we measure the advantage of an adversary in each sub-event. Finally, we will show that the sum of advantages is negligible due to the above propositions.

Theorem 1. TIM is secure in the sense that an adversary cannot generate any streaming data packet which is valid but differ from the original data, and the security can be guaranteed by two propositions, Proposition 1 and Proposition 2.

Proof 1. Let $params = \langle p, q, \alpha, H, G \rangle$ be set of the system public parameters for the target system. Let X be the long-term public key and x be the corresponding private key. Let $(\bar{y}, \bar{Y})$ be the set of long-term trapdoor key and hash key pair such that $\bar{Y} = \alpha^{\bar{y}} \bmod p$ for $\bar{y} \in \mathbb{Z}_q^*$. Let F be the data which will be attacked by an adversary.

Let E be the event that an adversary $\mathcal{A}$ succeeds in breaking the proposed scheme by generating a valid data packet and the forged data is a fresh packet in the sense that the same data packet was not produced ever before. Let $Adv_A(Event)$ be the advantage of the adversary $\mathcal{A}$ with respect to the event $Event$. Then, the adversary's ability of breaking our scheme can be defined as $Adv_A(E)$ since E is defined as the event that covers all attack scenarios. Note that for two disjoint events S_1 and S_2, we have $Adv_A(S_1 \cup S_2) \le Adv_A(S_1) + Adv_A(S_2)$ for any adversary $\mathcal{A}$. Suppose that the adversary $\mathcal{A}$ generates a forged data packet for j-th block such that $i = \lfloor j/k \rfloor$ and $\ell = j - ik$ which means that the block belongs to i-th MHT and the block is the ℓ-th block among k blocks in i-th MHT. The authentication information for the message block m_j is then $AI_j = \langle z_{ik,(i+1)k-1}, wit_j, Y_i \rangle$ where

- $z_{ik,(i+1)k-1}$ is a collision for the trapdoor hash function which satisfied the following relation: $TH_{Y_i}(h_{RH_{ik,(i+1)k-1}, Y_i}, z_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v, \bar{Y}}, r_v)$,
- wit_j is the sibling path information for m_j regarding i-th MHT,
- and Y_i is the i-th ephemeral public key.

Let $\{m_j, \langle z_{ik,(i+1)k-1}, wit_j, Y_i \rangle\}$ be the original data packet for j-*th* block. If the adversary succeeds in generating valid packet for j-*th* block, we should have $\{m_j^*, \langle z_{ik,(i+1)k-1}^*, wit_j^*, Y_i^* \rangle\}$ which is different from the original data packet. Let $RH_{ik,(i+1)k-1}^*$ be the root of the MHT computed from $\{m_j^*, wit_j^*\}$. Then we can define disjoint sub-events E_1, E_2, and E_3 according to the packet generated by the adversary. At first, we can consider two branches based on the conditions that the adversary can use forged $\langle f^*, t^*, r^* \rangle$ or not. If receivers already possess the correct tuple $\langle f_v, t_v, r_v \rangle$ or the forged tuple $\langle f^*, t^*, r^* \rangle$ is not valid, the adversary cannot use the forged tuple $\langle f^*, t^*, r^* \rangle$. In this case, we still can consider two branches based on the equality of $RH_{ik,(i+1)k-1}^*$ and $RH_{ik,(i+1)k-1}$. We define E_1 as the sub-event that two values are identical and E_2 as the sub-event that two values are different. Then, two sub-events can be simplified into a set of numerical conditions as followings:

- sub-event E_1. Adversary cannot use forged $\langle f^*, t^*, r^* \rangle$ and $RH^*_{ik,(i+1)k-1} = RH_{ik,(i+1)k-1}$.
- sub-event E_2. Adversary cannot use forged $\langle f^*, t^*, r^* \rangle$ and $RH^*_{ik,(i+1)k-1} \neq RH_{ik,(i+1)k-1}$.

For sub-event E_1 and E_2, we can assume that $wit^*_j \neq wit_j$ since the forged data packet should be used for authenticating a new message which is not identical with the original message. For the second sub-event, we can easily find the following additional condition

$$TH_{Y^*_i}(H(RH^*_{ik,(i+1)k-1} \| Y^*_i), z^*_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v).$$

It is trivial that the equation should hold unless $RH^*_{ik,(i+1)k-1}$ is regarded as invalid. To cover all event E, we need to define one more sub-event E_3 to cover the case where the adversary can generate and user forged $\langle f^*, t^*, r^* \rangle$ which is not equal to the original value, i.e., $\langle f^*, t^*, r^* \rangle \neq \langle f_v, t_v, r_v \rangle$. The last sub-event can be simplified as following:

- sub-event E_3. Adversary can use forged $\langle f^*, t^*, r^* \rangle$.

The last sub-event implies that the forged data packet should hold the following conditon $TH_{Y^*_i}(H(RH^*_{ik,(i+1)k-1} \| Y^*_i), z^*_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f^*_v,\bar{Y}}, r^*_v)$ since the forged $\langle f^*, t^*, r^* \rangle$ can be used as a part of verification information only if it holds the condition. Until now, we define three sub-events whose disjoint union cover E. From now, we show that an adversary cannot generate fogery which belongs to any sub-event due to the security of underlying schemes.

In the first sub-event E_1, we have two conditions $wit^*_j \neq wit_j$ and $RH^*_{ik,(i+1)k-1} = RH_{ik,(i+1)k-1}$ at the same time. The conditions imply that there exist two values whose hash value is same, and it also implies that the adversary succeeds in finding a collision of a hash function which is used for the MHT. As a result, if the manipulated data packet belongs to E_1, we can say that the security of the proposed scheme can be reduced to the security of the underlying hash function in terms of the collision resistance. Hence, we have $Adv_A(E_1) \leq e_H$ where e_H is the advantage of an adversary who breaks the hash function in terms of the collision resistance. If the underlying hash function is hard to break, the advantage e_H is negligible. In the second sub-event E_2, a new root is used as a part of authenticating information, i.e., we have $RH^*_{ik,(i+1)k-1} \neq RH_{ik,(i+1)k-1}$. To be used as a part of authenticating information, it should satisfy $TH_{Y^*_i}(H(RH^*_{ik,(i+1)k-1} \| Y^*_i), z^*_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$. Since the original information satisfies $TH_{Y_i}(H(RH_{ik,(i+1)k-1} \| Y_i), z_{ik,(i+1)k-1}) = TH_{\bar{Y}}(h_{f_v,\bar{Y}}, r_v)$, we have

$$TH_{Y_i}(H(RH_{ik,(i+1)k-1} \| Y_i), z_{ik,(i+1)k-1}) = TH_{Y^*_i}(H(RH^*_{ik,(i+1)k-1} \| Y^*_i), z^*_{ik,(i+1)k-1}).$$

Then, we can see that the forged packet belongs to the second type implies the existence of a collision for the underlying trapdoor commitment scheme. Hence, the advantage of $\mathcal{A}$ in the second event is bounded by the advantage of an adversary against the trapdoor commitment scheme in terms of the trapdoor collision resistance, and thus we have $Adv_A(E_2) \leq e_{TC}$ where e_{TC} is the advantage of an adversary against the trapdoor commitment scheme. The upper bound e_{TC} is negligible if it is hard to break the underlying trapdoor commitment scheme in terms of the trapdoor collision resistance whose hardness is identical with the hardness of the discrete logarithm problem. Since the discrete logarithm problem is well-known hard problem, we can see that $Adv_A(E_2)$ is negligible. Finally, if the forged packet belongs to the last sub-event, the tuple $\langle f^*, t^*, r^* \rangle$ is a valid signature on a fresh message $TH_{\bar{Y}}(h_{f^*_v,\bar{Y}}, r^*_v)$, which

implies that the underlying signature scheme is broken in terms of existential unforgeability since a new valid signature is included in the forged data packet. Hence, we have $Adv_A(E_3) \le e_S$ where e_S is the advantage of an adversary against the underlying signature scheme. Since we adopt a secure scheme whose security already proven in the literature, we can see that $Adv_A(E_3)$ is negligible.

As we explained in the beginning of the proof, the adversary $\mathcal{A}$ succeeds in breaking the proposed scheme if it can generate a valid data packet. The advantage of the adversray is defined as $Adv_A(E)$ and it can be rewritten as $Adv_A(E) \le Adv_A(E_1) + Adv_A(E_2) + Adv_A(E_3)$ since E_1, E_2, and E_3 are disjoint sub-events of E such that $E = E_1 \bigcup E_2 \bigcup E_3$. Hence, we have $Adv_A(E) \le e_H + e_{TC} + e_S$. Due to two propositions, we can see that $Adv_A(E)$ is negligible. In other works, the adversary's advantage in generating valid forgery for each sub-event is negligible due to the security of underlying security tools. □

We have also verified the security of the proposed scheme with AVISPA [27] which is an automated formal security analysis tool. We have simplified the proposed streaming authentication scheme in order to be run in AVISPA and we have used CAS+ language to describe the proposed scheme for AVISPA. As the result of backend(OFMC and CL-AtSe) analysis in AVISPA, our scheme turns out to be safe.

4.2 Computational Overhead Analysis

Table 2. Notations for Comparisons

Symbols	Definitions
$\mathbf{H}$	Hash operation such as SHA-256
$\mathbf{I_q}$, $\mathbf{M_q}$, $\mathbf{A_q}$	Modular inversion, multiplication, and addition over prime q
$\mathbf{M_p}$	Modular multiplication over prime p
$\mathbf{S}$	Signing operation in DSA, requiring ($\mathbf{E_D} + \mathbf{I_q} + 2\mathbf{M_q} + \mathbf{A_q} + \mathbf{H}$)
$\mathbf{V}$	Verifying operation in DSA, requiring ($2\mathbf{E_D} + \mathbf{M_p} + \mathbf{I_q} + 2\mathbf{M_q} + \mathbf{H}$)
$\mathbf{E_D}$	DSA exponentiation operation such that $((B^y \bmod p) \bmod q)$, where B is an element of order q in $\mathbb{Z}_p^*$, $y \in_R \mathbb{Z}_q^*$, p and q are 1,024-bit and 160-bit primes

In this subsection, we analyze theoretical overhead of TIM. **Table 2** shows the notations used for analyzing TIM's performance and performance comparison with previously presented schemes. Following theorem proves the computation costs for signing and verification in TIM.

Theorem 2. For authenticating a segment consisting of n data blocks ($n = l \cdot k$), offline signing requires $(\mathbf{E_D} + 2\mathbf{M_q} + 2\mathbf{A_q} + 2\mathbf{H})$ and online signing requires $(l \cdot (2k\mathbf{H} + \mathbf{M_q} + 2\mathbf{A_q}))$, respectively.

Proof 2. Step 6 and Step 7 of offline signing in **Alg. 2** requires ($\mathbf{E_D} + \mathbf{M_q} + \mathbf{A_q} + \mathbf{H}$) and ($\mathbf{M_q} + \mathbf{A_q} + \mathbf{H}$), respectively. Thus, offline signing process requires ($\mathbf{E_D} + 2\mathbf{M_q} + 2\mathbf{A_q} + 2\mathbf{H}$) in total and it is executed once per segment. Online signing in **Alg. 3** consists of MHT construction and collision value computation. Constructing MHT composed of k data blocks requires $(2k-1)\mathbf{H}$. If k is even and 2's exponentiation value like $k = 2^a$, then k is $2^{log_2 k}$. To construct a MHT of k data blocks, at first data blocks' hash values need to be computed and

these are leaf hash values of the MHT, which requires $2^a\mathbf{H}$. Then, intermediate hash nodes are computed until the root hash value is computed. Thus, we can calculate the number of hash operations for constructing a MHT of k data blocks as $2k-1=(2^{a+1}-1)$. Step 6 in **Alg. 3** requires $(\mathbf{M_q}+2\mathbf{A_q}+2k\mathbf{H})$ including both root hash value computation and collision value computation. Online signing process is repeated l times to generate n data blocks' authentication information where $n=k\cdot l$. Thus, the total cost of online signing for generating authentication information of n data blocks in a segment is $l\cdot\{\mathbf{M_q}+2\mathbf{A_q}+2k\mathbf{H}\}$.

Theorem 3. For authenticating a segment consisting of n data blocks ($n=l\cdot k$), the signature verification and verifying n data blocks requires $(4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})$ and $l\cdot(2\mathbf{E_D}+\mathbf{M_p}+(klog_2k+k+1)\mathbf{H})$, respectively.

Proof 3. To proceed data block verification, firstly a signature of each segment needs to be verified. In **Alg. 4**, Step 2 computing trapdoor hash value requires $(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H})$ and Step 3 executing DL-Schnorr signature verification consumes $(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H})$, respectively. Thus, signature verification process requires $(4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})$ in total and it is executed only once per a segment. Data block verification consists of MHT-based verification and THF-based verification. THF-based verification in Step 16-28 of **Alg. 5** requires a trapdoor hash value computation which consumes $(2\mathbf{E_D}+\mathbf{M_p}+(log_2k+2)\mathbf{H})$. In the MHT-based verification, the validity of received data block is checked until the root hash value of the MHT is computed while searching the computed hash values in the AHT. This process requires $(log_2k+1)\mathbf{H}$ at the worst case because the number of hash operation is identical to MHT's height. THF-based verification is executed for the first received data block among k data blocks of the MHT and MHT-based verification is applied to remaining $k-1$ data blocks. Thus, the total cost for authenticating k data blocks in a MHT is $(2\mathbf{E_D}+\mathbf{M_p}+(klog_2k+k+1)\mathbf{H})$. Since there are l MHT blocks in a segment, the total cost for authenticating data blocks in a segment is $((4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})+l\cdot(2\mathbf{E_D}+\mathbf{M_p}+(klog_2k+k+1)\mathbf{H})$.

4.3 Computational Overhead Comparison

In this subsection, we compare TIM with previously proposed other schemes with regard to computational overhead. For fair comparison, we assume that MABS [9] and WL [4] use DSA as their underlying signature scheme for fair comparison.

Table 3 compares TIM with previously presented mechanisms with respect to latency (buffering) and computational overhead at sender and receiver sides. We assume that all mechanisms in **Table 3** try to authenticate a segment consisting of n data blocks. In addition, for fair comparison we assume that the size of MHT used in WL and TIM is the same as the size for batch verification in MABS and DL-SA as k size. The latency column denotes that the sender or the receiver needs to buffer how many data blocks to generate or to verify authentication information.

Since MABS-B and MABS-E generate independent signature for each data block, they require n $(\mathbf{E_D}+\mathbf{I_q}+2\mathbf{M_q}+\mathbf{A_q}+\mathbf{H})$ for signing n data blocks. For efficiency of verification, MABS-B and MABS-E utilize batch verification for k buffered signatures. Thus, they requires $(2\mathbf{E_D}+k(\mathbf{I_q}+2\mathbf{M_p}+3\mathbf{M_q}+\mathbf{H}))$ rather than $k\mathbf{V}$. Since MABS-E equipped with MHT is the

enhanced version of MABS-B with respect to DoS defense, it requires more hash computations than MABS-B at both sender and receiver sides.

Table 3. Computational overhead comparison with previously presented schemes (We assume that a segment consists of $n\ (=k\cdot l)$ blocks).

Method	Sides	Latency	Computational overhead
MABS-B	*Sender*	1	$n\ (\mathbf{E_D}+\mathbf{I_q}+2\mathbf{M_q}+\mathbf{A_q}+\mathbf{H})$
	Receiver	k	$l\cdot(2\mathbf{E_D}+k(\mathbf{I_q}+2\mathbf{M_p}+3\mathbf{M_q}+\mathbf{H}))$
MABS-E	*Sender*	k	$l\cdot(k(\mathbf{E_D}+\mathbf{I_q}+2\mathbf{M_q}+\mathbf{A_q}+\mathbf{H})+(2k-1)\mathbf{H})$
	Receiver	k	$l\cdot(2\mathbf{E_D}+k(\mathbf{I_q}+2\mathbf{M_p}+3\mathbf{M_q}+(log_2k+1)\mathbf{H}))$
WL	*Sender*	k	$l\cdot((\mathbf{E_D}+\mathbf{I_q}+2\mathbf{M_q}+\mathbf{A_q}+\mathbf{H})+(2k-1)\mathbf{H})$
	Receiver	1	$l\cdot((2\mathbf{E_D}+\mathbf{M_p}+\mathbf{I_q}+2\mathbf{M_q}+\mathbf{H})+k(log_2k+1)\mathbf{H})$
DL-SA-B	*Sender*	1	$((\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})+(n-1)(\mathbf{M_q}+2\mathbf{A_q}+\mathbf{H}))$
	Receiver	1	$((4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})+(n-1)(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H}))$
DL-SA-E	*Sender*	1	$((\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})+(n-1)(\mathbf{M_q}+2\mathbf{A_q}+\mathbf{H}))$
	Receiver	k	$((4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})+(l-1)((k+1)(\mathbf{E_D}+\mathbf{M_p})+k(\mathbf{A_q}+\mathbf{H}))+(k(\mathbf{E_D}+\mathbf{M_p})+(k-1)(\mathbf{A_q}+\mathbf{H})))$
TIM Online	*Sender*	k	$l\cdot(\mathbf{M_q}+2\mathbf{A_q}+2k\mathbf{H})$
	Receiver	1	$((4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})+l\cdot(2\mathbf{E_D}+\mathbf{M_p}+(klog_2k+k+1)\mathbf{H}))$

WL applies MHT-based signature generation to k buffered data blocks. Since single MHT-based signature generation for k blocks requires $(\mathbf{S}+(2k-1)\mathbf{H})$, the total cost of generating authentication information for n blocks is $l\cdot(\mathbf{S}+(2k-1)\mathbf{H})$. For verifying k data blocks, WL applies signature verification to the first received block and applies hash value comparisons to next $(k-1)$ blocks as computing the root hash value of the MHT. Thus, WL requires $(\mathbf{V}+k(log_2k+1)\mathbf{H})$ for verifying k data blocks.

DL-SA-B is a trapdoor hash function-based stream authentication method. Its signing process consists of initial block signing for the first data block and subsequent block signing for the next $(n-1)$ blocks. The initial block signing requires $(\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})$ and subsequent block signing consumes $(\mathbf{M_q}+2\mathbf{A_q}+\mathbf{H})$. Thus, DL-SA-B consumes $((\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})+(n-1)(\mathbf{M_q}+2\mathbf{A_q}+\mathbf{H}))$ to generate authentication information for n blocks[3]. Since DL-SA-B requires $(4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})$ and $(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H})$ to verify the initial block and each subsequent block, respectively, the total cost for verifying n data blocks is $((4\mathbf{E_D}+2\mathbf{M_p}+2\mathbf{H})+(n-1)(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H}))$. Namely, DL-SA-B requires at least $2\mathbf{E_D}$ to verify each data block and this might be huge computational burden for smart devices like smart phones, smart pads, and so on. DL-SA-E applies batch verification that requires data block buffering at the receiver-side in order to enhance the verification performance. DL-SA-E could reduce the overhead for verifying k data blocks from $k(2\mathbf{E_D}+\mathbf{M_p}+\mathbf{H})$ to $((k+1)(\mathbf{E_D}+\mathbf{M_p})+k(\mathbf{A_q}+\mathbf{H}))$. Even though DL-SA-E can reduce the cost for verification

[3] When estimating the signing overhead in DL-SA-B and DL-SA-E, we omit the overhead for computing an ephemeral public key in initial block signing and an ephemeral hash key in subsequent block signing since they can be precomputed.

roughly about in half, it still requires at least $\mathbf{E_D}$ for verifying each block.

TIM makes use of the advantages of MHT and THF simultaneously. TIM requires $((\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})+l\cdot(\mathbf{M_q}+2\mathbf{A_q}+2k\mathbf{H}))$ to generate authentication information for n data blocks[4]. Since $(\mathbf{E_D}+2\mathbf{M_q}+2\mathbf{A_q}+2\mathbf{H})$ is the overhead from offline signing, it can be conducted at the offline phase before sending stream data. Thus, TIM consumes only $l\cdot(\mathbf{M_q}+2\mathbf{A_q}+2k\mathbf{H}))$ when transmitting a segment consisting of n data blocks in stream data, which is much more efficient than WL in terms of signing cost. Furthermore, since online signing in TIM computes a collision value per a MHT block consisting of k data blocks, the number of $\mathbf{M_q}$ in TIM's online signing process is reduced by about $(1/k)$ compared with DL-SA-B and DL-SA-E which compute a collision value per a data block. Actually, even though the number of $\mathbf{H}$ is almost doubled in TIM's online signing process compared to signing process in DL-SA-B and DL-SA-E, the overhead for $\mathbf{H}$ is much lower than that for $\mathbf{M_q}$. Since the verification process in TIM utilizes the advantage of MHT, its cost is much lighter than those of DL-SA-B and DL-SA-E by replacing heavy trapdoor hash value computation with efficient hash comparisons. Namely, its cost for verifying n data blocks is about $(1/k)$ of DL-SA-B's verification cost and $\left(2/(k+1)\right)$ of DL-SA-E's verification cost, respectively. In summary, TIM not only provides more efficient signing performance than WL, DL-SA-B and DL-SA-E, but also sharply reduces the overhead for verification by utilizing MHT's property compared with DL-SA-B and DL-SA-E.

Table 4. Transmission overhead comparison with other schemes
($|\cdot|$ symbol denotes the output size of corresponding operation result. $|r|=|s|=|u|=|w|=|q|$.).

Method	Transmission overhead
MABS-B	$n(\|r\|+\|s\|)$
MABS-E	$l\cdot k(\|r\|+\|s\|+\log_2 k\|\mathbf{H}\|)$
WL	$l\cdot k(\|r\|+\|s\|+\log_2 k\|\mathbf{H}\|)$
DL-SA-B, DL-SA-E	$\|u\|+(n-1)\|w\|$
TIM	$\|t_v\|+l\cdot k(\|z_{ik,(i+1)k-1}\|+\log_2 k\|\mathbf{H}\|)$

Table 4 compares TIM with previously presented schemes with respect to transmission overhead. Since MABS-B, MABS-E, and WL make use of DSA as their underlying signature scheme, the signature consists of r and s which are values over prime q. Since MABS-E, WL, and TIM use MHT, they send witness information consisting of $(\log_2 k|\mathbf{H}|)$. However, this overhead can be alleviated by applying protocol level approaches [21]. For example, firstly sending complete *witness* for computing root hash value of the MHT, then sending related messages can reduce the number of redundant witness information transmission. In DL-SA-B and DL-SA-E, u is the signature of the initial data block and each of subsequent data blocks requires w which is a collision value. u and w are values over prime q in DL-SA-B and DL-SA-E. In TIM, t_v is the signature of the v-th segment and each data block in the segment requires the transmission of collision value $|z_{ik,(i+1)k-1}|$ and witness information consisting of

[4] We omit the overhead for generating key materials since they can be computed at offline phase similar to DL-SA-B and DL-SA-E.

$\log_2 k \,|\mathbf{H}|$. Even though DL-SA-B, DL-SA-E, and TIM requires ephemeral public key and hash keys for verifying data blocks, we omit the overhead for transmitting these keys since these keys needs to be transmitted only once to receivers and they can be reused.

5. Experimental Evaluation

Table 5. Block signing time in TIM (The figures are average single data block signing timings for 100-MByte stream data. The time for signing a data block in DL-SA-E is 0.000120026 ms. #data blocks means the size of MHT in TIM.).

#data blocks	TIM	Improvement
2	0.0000610323 ms	1.97 times
4	0.0000315721 ms	3.80 times
8	0.0000166578 ms	7.21 times
16	0.0000090785 ms	13.22 times
32	0.0000053429 ms	22.46 times

Table 6. Verification timing comparison
(The figures are average single data block verification timings for 100-MByte stream data.).

#data blocks	DL-SA-E	TIM	Improvement
2	0.119877 ms	0.07415 ms	1.62 times
4	0.096436 ms	0.03689 ms	2.61 times
8	0.084070 ms	0.01871 ms	4.49 times
16	0.078259 ms	0.00936 ms	8.36 times
32	0.075015 ms	0.00465 ms	16.14 times

We have implemented TIM and DL-SA-E on Ubuntu Linux 11.04 on a Desktop PC with Intel i7 3.40GHz quadcores CPU by using OpenSSL-1.0.1d version for timing comparison. Since OpenSSL-1.0.1d does not support LL exponentiation method [10] that is a kind of fixed-base exponentiation algorithm, we have implemented the LL method for performance enhancement. We have measured time for transmitting about 100-MByte stream data from a sender to a receiver (It consists of 3,200 segments and each segment contains 32 blocks of 1024-byte.[5]). **Table 5** shows the timing for generating single data block's authentication information with online signing in TIM as increasing the size of MHT from 2 to 32. Note that we have implemented DL-SA-E and measured the timing for generating a data block's authentication information as 0.000120026 ms. In DL-SA-E, the signing cost is constant as about $(\mathbf{M_q}+\mathbf{H})$ per data block regardless of MHT size. The signing cost in TIM depends on the applied MHT size k as about $((1/\mathrm{k})\mathbf{M_q}+(2/k)\mathbf{A_q}+2\mathbf{H})$. **Table 5** shows that the online signing in TIM provides much improved performance compared with DL-SA-E since it can reduce the number of collision value computation from one per each data block to one per k data blocks.

Table 6 shows the verification timing of TIM and DL-SA-E by increasing k to 2, 4, 8, 16, and 32. In case of DL-SA-E, we have applied batch verification scheme and simultaneous multi-exponentiation method according to [19]. With respect to the theoretical computation overhead, DL-SA-E needs $\{(k+1)(\mathbf{E_D}+\mathbf{M_p})+k(\mathbf{A_q}+\mathbf{H})\}$ for verifying k data blocks by

[5] As in [19], public hash keys Y_i for all i where $1 \le i \le 32$ are transmitted in the first segment with additional 1,024-bit per block and not transmitted for subsequent segments.

using k size of batch verification method, while TIM requires $(2\mathbf{E_D} + \mathbf{M_p} + (klog_2k + k + 1)\mathbf{H})$. Since the overhead for $\mathbf{H}$, $\mathbf{A_q}$ and $\mathbf{M_p}$ is negligible compared with that for $\mathbf{E_D}$, the theoretical performance improvement of TIM against DL-SA-E can be computed as $(k+1)/2$. For each k = 2, 4, 8, 16, and 32, experimental results show that TIM achieve 1.62, 2.61, 4.49, 8.36, and 16.14 times improvement compared with DL-SA-E using k size of batch verification method. Moreover, it is noticeable that this experimental improvement ratio obtained by replacing k with actual value is similar to the theoretical improvement ratio. It proves the effectiveness of TIM in view of both experimental and theoretical point.

6. Concluding Remarks

In this paper, we have proposed TIM, an enhanced trapdoor hash function-based stream authentication mechanism integrated with the Merkle Hash Tree, which can significantly reduce the overhead for signing at sender-side and verification at receiver-side, respectively. By using the property of the THF, TIM achieves the optimized signing cost and it outperforms typical MHT-based authentication mechanism such as WL and previously proposed trapdoor hash function-based mechanism DL-SA. TIM integrates MHT property into the trapdoor hash function-based mechanism with a novel manner, which results in noticeably reducing the verification overhead compared with DL-SA. We have analyzed the correctness and theoretical overhead of TIM. Both theoretical overhead analysis and experimental evaluation show that TIM outperforms previously proposed deterministic authentication mechanisms. We expect that TIM can be utilized from servers to mobile devices including sensors, smartcards, smartphones, and so on due to its performance efficiency.

References

[1] R. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Proc. of Crypto'87*, pp. 369-378, 1987. Article (CrossRef Link).
[2] R. Merkle, "A Certified Digital Signature," in *Proc. of Crypto'89, LNCS 435*, pp. 218-238, 1989. Article (CrossRef Link).
[3] C.-P. Schnorr, "Efficient Signature Generation by Smart Cards," *J. Cryptology*, Vol. 4, No. 3, pp. 161-174, 1991. Article (CrossRef Link).
[4] C.K. Wong and S.S. Lam, "Digital Signatures for Flows and Multicasts," *IEEE/ACM Trans. Networking*, Vol. 7, No. 4, pp. 502-513, 1999. Article (CrossRef Link).
[5] A. Perrig, R. Canetti, J.D. Tygar, and D.X. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," *IEEE Symposium of Security and Privacy*, pp. 56-73, 2000. Article (CrossRef Link).
[6] P. Golle and N. Modadugu, "Authenticating Streamed Data in the Presence of Random Packet Loss," in *Proc. of Proc. Network and Distributed System Security Symp. (NDSS)*, 2001.
[7] J.M. Park, E.K.P. Chong, and H.J. Siegel, "Efficient Multicast Stream Authentication using Erasure Codes," *ACM Trans. Information and System Security*, vol. 6, no. 2, pp. 258-285, 2003. Article (CrossRef Link).
[8] K. Ren, W. Lou, K. Zeng, and P.J. Moran, "On Broadcast Authentication in Wireless Sensort Networks," *IEEE Transactions on Wireless Communications*, Vol. 6, No. 11, pp. 4136-4144, 2007. Article (CrossRef Link).
[9] Y. Zhou, X. Zhu, and Y. Fang, "MABS: Multicast Authentication Based on Batch Signature," *IEEE Transactions on Mobile Computing*, Vol. 9, No. 7, pp. 982-993, 2010. Article (CrossRef Link).

[10] C.H. Lim and P.J. Lee, "More Flexible Exponentiation with Precomputation," in *Proc. of Crypto'94, LNCS839*, pp. 95-107, 1994. Article (CrossRef Link).
[11] S. Even, O. Goldreich, and S. Micali, "Online/Offline Digital Schemes," in *Proc. of Crypto'89, LNCS 435*, pp. 263-275, 1989. Article (CrossRef Link).
[12] Hugo Krawczyk and Tal Rabin, "Chameleon Signatures," in *Proc. of Symposium on Network and Distributed Systems Security(NDSS'00)*, pp. 143-154, 2000.
[13] A. Shamir and Y. Tauman, "Improved Online/Offline Signature Schemes," in *Proc. of Crypto'01, LNCS 2139*, pp. 355-367, 2001. Article (CrossRef Link).
[14] X. Chen, F. Zhang, and K. Kim, "Chameleon Hashing without Key Exposure," in *Proc. of Seventh Int'l Conf. Information Security (ISC)*, pp. 87-98, 2004. Article (CrossRef Link).
[15] G. Ateniese and B. de Medeiros, "On the Key Exposure Problem in Chameleon Hashes," in *Proc. of Fourth Int'l Conf. Security in Comm. Networks (SCN)*, pp. 165-179, 2004. Article (CrossRef Link).
[16] M. Mehta and L. Harn, "Efficient One-Time Proxy Signatures," *IEE Proc. Comm.*, vol. 152, no. 2, pp. 129-133, Apr. 2005. Article (CrossRef Link).
[17] L. Harn, W.-J. Hsin, and C. Lin, "Efficient Online/Offline Signature Schemes Based on Multiple-Collision Trapdoor Hash Families," *The Computer J.*, vol. 53, no. 9, pp. 1478-1484, 2010. Article (CrossRef Link).
[18] S. Chandrasekhar, S. Cxhakrabarti, M. Singhal, and K.L. Calvert, "Efficient Proxy Signatures Based on Trapdoor Hash Functions," *IET Information Security, Speacial Issue on Multi-Agent and Distributed Information Security*, vol. 4, no. 4, pp. 322-332, 2010. Article (CrossRef Link).
[19] S. Chandrasekhar, S. Chakrabarti, and M. Singhal, "A Trapdoor Hash-Based Mechanism for Stream Authentication," *IEEE Transactions on Dependable and Secure Computing*, Vol.9, No.5, pp. 699-713, 2012. Article (CrossRef Link).
[20] S. Chandrasekhar and M. Singhal, "Multi-trapdoor Hash Functions and Their Applications in Network Security," in *Proc. of IEEE Conf. on Comm. and Network Security(CNS 2014)*, pp. 463-471, 2014. Article (CrossRef Link).
[21] W. Wong and M.F. Magalhaes, "Security Approaches for Information-Centric Networking," *Applied Cryptography and Network Security*, ISBN 978-953-51-0218-2, InTech, 2012. Article (CrossRef Link).
[22] Robert H. Deng and Yanjiang Yang, "Achieving End-to-End Authentication in Intermediary-Enabled Multimedia Delivery Systems," in *Proc. of ISPEC 2007, LNCS 4464*, pp. 284-300, 2007. Article (CrossRef Link).
[23] Yi Sun, Xingyuan chen, and Xuehui Du, "An Efficient Elliptic Curve Discrete Logarithm based Trapdoor Hash Scheme without Key Exposure," *Journal of Computers*, Vol. 8, No. 11, pp. 2851-2856, 2013. Article (CrossRef Link).
[24] S. Chandrasekhar and M. Singhal, "Efficient and Scalable Aggregate Signcryption Scheme Based on Multi-trapdoor Hash Functions," in *Proc. of IEEE Conference on Communications and Network Security (CNS)*, pp. 610-618, 2015. Article (CrossRef Link).
[25] S. Chandrasekhar and M. Singhal, "Efficient and Scalable Query Authentication for Cloud-based Storage Systems with Multiple Data Source," *IEEE Transactions on Services Computing*, pp. 520-533, 2017. Article (CrossRef Link).
[26] T.-Y. Youn, S. Lee, S. H. Hong, and Y.-H. Park, "Practical RSA-PAKE for Low-power Device in Imbalanced Wireless Networks," *International Journal of Distributed Sensor Networks*, Volume 2014, Article ID 125309, 6 pages, 2014. Article (CrossRef Link).
[27] Armando A., Basin D., Boichut Y., "The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications," *Computer Aided Verification. CAV 2005, LNCS 3576*, pp. 281-285, 2005. Article (CrossRef Link).

Dr. Seog Chung Seo received the B.S. in Information & Computer Engineering from Ajou University, Suwon, Korea and the M.S. in Information and Communications from Gwangju Institute of Science and Technology (GIST), Gwangju, Korea in 2005 and 2007, respectively. He received the Ph.D. degree at Korea University, Seoul, Korea in 2011. He worked as a research staff member at SAIT (Samsung Advanced Institute of Technology) and Samsung DMC R&D Center, from Sep. 2011 to April 2014. He currently works for the affiliated institute of ETRI in Korea since May 2014. His research interests include public-key cryptography, its efficient implementations on various IT devices, cryptographic module validation program (CMVP), network security, and data authentication algorithms.

Dr. Taek-Young Youn received his BS, MS, and Ph.D from Korea University in 2003, 2005, and 2009, respectively. He is currently a senior researcher at Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. From 2016, he serve as an associate professor in University of Science and Technology (UST), Daejeon, Korea. His research interests include cryptography, information security, authentication, data privacy, and security issues in various communications.