applied sciences

MDPI

Article

The Shortest Verification Path of the MHT Scheme for Verifying Distributed Data

Daeyoub Kim [1] and Jihoon Lee [2,*]

[1] Department of Information Security, Suwon University, Hwaseong-si 18328, Gyeonggi-do, Korea
[2] Department of Smart Information Communication, Sangmyung University, Cheonan-si 31066, Chungcheongnam-do, Korea
* Correspondence: vincent@smu.ac.kr

Abstract: One of the most common approaches for enhancing network performance is to retrieve data from nearby data holders that have previously obtained the desired data, not only from the original data source itself. In this case, since a data receiver cannot identify a practical data sender, it is necessary to verify both the received data and the data sender. Moreover, a data sender generally fragments the data into several small segments and sends them. Therefore, if these segments are retrieved from multiple unknown senders, the receiver must verify every segment to safely use the data. MHT (Merkle hash tree) is suitable for efficiently verifying the set of segments shared in the network. NDN (named-data networking) and Bitcoin utilize MHT to verify transmitted data. However, a data authentication scheme based on the MHT has an inefficient factor that repeatedly computes the same node values of the MHT and are repeatedly computed. The larger the size of the MHT is, the greater the number of calculation iterations. Therefore, as a result, the authentication scheme's inefficiency is also more severe. When a sender transmits data consisting of many segments through NDN, the data authentication time may take longer than the data transmission time. Hence, in this paper, the degree of the MHT's inefficiency and the pattern of the iterated operation of the MHT are analyzed first. The proposed improvement is to find repeatedly used node values, store them internally, and use the stored node values without recalculation when required to reuse them. For that process, a rule to select such node values is given. Additionally, when verifying the leaf node value of the MHT, the MHT-based authentication scheme asks a verifier to compute all node values on the path from the leaf node to the root node of the MHT. This paper demonstrates the proposed shortest path selection for verifying the leaf node value. The proposed scheme, using saved node values and the shortest path, reduces the computational overhead of the MHT and improves service latency. It has been proven from performance evaluations that the proposed scheme decreases the computational overhead by more than one-third if the number of segments is more than 1024.

Keywords: distributed network; future internet; MEC; data authentication; MHT

check for updates

Citation: Kim, D.; Lee, J. The Shortest Verification Path of the MHT Scheme for Verifying Distributed Data. *Appl. Sci.* **2022**, *12*, 11194. https://doi.org/10.3390/app122111194

Academic Editor: Giacomo Fiumara

Received: 12 September 2022
Accepted: 2 November 2022
Published: 4 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the explosive usage growth of intelligent connected devices and the Internet of Things (IoT) devices, such devices inevitably generate large amounts of data transmitted over networks. Since data traffic is increasing rapidly, content/network service providers must continuously install additional network facilities to keep up with such traffic growth [1]. Therefore, providers seek alternative solutions such as P2P (peer-to-peer network) and CDN (content delivery network) rather than just installing more facilities to handle data traffic increases. The typical approach of CDN and P2P is that users retrieve data from nearby and multiple sources that cache the data [2,3]. Similar to CDN/P2P, a general approach for enhancing network performance is to retrieve data from nearby data holders that have previously obtained the data, not only from the original data source itself. The ICN (information-centric networking architecture), a future Internet architecture,

follows such an approach to utilizing in-network caches [4,5]. The NDN (named-data networking architecture) is one of the ICNs [6,7]. The NDN makes network nodes such as network routers cache transmitted data. The network nodes caching the data directly respond to requests for the data as well as forward the requests to other nodes. Multi-access edge computing (MEC) architecture also follows such an approach [8]. The common factor of P2P, ICN/NDN, and MEC is that a data requestor can receive data from multiple nodes that have cached the requested data. In these cases, the problem is that the requestor cannot identify the cache nodes because the status of cached data in nodes is continuously changed. Especially, ICN/NDN does not provide the information of the cache nodes to end-users because of security concerns. Since the requestor receives the data from multiple unspecified and unknown nodes, it cannot verify whether attackers polluting the data are among the nodes. Hence, when using such technologies, the receiver must verify the received data.

When using advanced network technologies such as ICN, one of the critical issues is how to authenticate received data efficiently. A data authentication scheme has been proposed in various ways because a data authentication scheme must consider service characteristics [9,10]. A general method to authenticate data is to verify the digital signature of the data generated by the publisher of the data. A mechanism authenticating the signer's public key, such as PKI, is essential when verifying a digital signature. Additionally, a digital signature/verification scheme requires high computation overhead. It may not be suitable for ICN/NDN to utilize a digital signature/verification scheme in its original form to authenticate transmitted data. The ICN/NDN, as technologies of a transport/network layer, generally fragment the data into several small segments and send the segments. That is, they regard each data segment as a network traffic unit. If they utilize a digital signature/verification scheme, a data publisher must generate a digital signature for each segment of the data, and a data receiver must verify all signatures for using the data. However, such a process can cause long service delays.

MHT (Merkle hash tree) is a valuable scheme to authenticate a data set [11]. Various services, such as blockchains [12–15] and outsourcing services [16,17], utilize MHT to verify a data set. The NDN also utilize MHT to authenticate segment sets. Some NDN applications such as connected vehicles, smart homes, and IoT-based disaster management need to share information as quickly as possible [18–21]. However, the MHT-based data authentication scheme is still the major cause of NDN inefficiency [22]. Therefore, alternatives have been proposed [23–25]. There are two reasons for the inefficiency of MHT-based data authentication. First, the MHT must repeatedly compute the node values of the same internal nodes of its binary tree. Since NDN assigns each fragment of data to a leaf node of MHT, the binary tree size of MHT increases proportionally to the number of fragments of data transmitted. This means that the greater the number of fragments of the data being transferred, the more severe the degree of repeated operations can become. Second, it must verify a digital signature of the root node value for authenticating each leaf node value [24]. Such an iterated digital signature verification also causes serious inefficiencies.

Our Contributions. The efficiency of the MHT is improved in this paper. The iterated operations for computing node values of nodes on a full verification path, from a leaf node to the root node, cause such inefficiency. Hence, first, when verifying a leaf node value, a verifier saves some valid internal node values and reuses them without recomputing them when verifying other leaf node values. For that task, the process for identifying valid node values that a verifier saves for each leaf node is proposed.

Moreover, the concept of the *shortest verification node path* for verifying a leaf node value is demonstrated. If a user has saved valid node values while verifying a leaf node value, the user does not need to confirm a leaf node's full verification path when verifying the next-order leaf node value. Instead, the user can use another verification path shorter than the leaf node's full verification path. In this paper, an easy way to select the shortest verification path for each leaf node based on the index of the leaf node is proposed. By using the shortest verification path for each leaf node and saving valid node values, a verifier

can substantially reduce the number of node value calculations. This means reducing the practical service latency.

Additionally, we propose a method to minimize the number of digital signature verification procedures or to omit the digital signature verification procedures that in some cases do not require data creator verification. In this case, a verifier significantly reduces service latency. We demonstrate the proposed scheme's efficiency by evaluating NDN's computational overhead.

In the case of some applications/services using MHT, a fast authentication scheme is essentially needed. Connected cars and IoT-based disaster management are representative examples [18–21]. Since devices for such applications/services usually have enough memory, the proposed scheme is adaptable to such applications/services. Additionally, multimedia devices are also suitable for applying the proposed scheme because their memory specifications are generally sufficient. Generally, as user/system devices may have different specifications and services' requirements may differ, computational capabilities or the memory limit of such devices may vary accordingly. Hence, service providers can selectively utilize the proposed scheme according to the characteristics of such services/devices.

Organization. The rest of this paper is organized as follows: Section 2 contains the description of the MHT and the analysis its performance. The proposed method is then presented in Section 3, and the evaluation results are shown in Section 4. Finally, we conclude our work in Section 5.

2. MHT for Data Authentication

This section describes the structure of the MHT and the operation of the MHT for verifying data segments. For a better understanding, when describing MHT operation and evaluation, NDN, which uses MHT to verify data segments in practice, is utilized.

2.1. MHT Overview

To efficiently transmit large-size data, a data publisher usually divides the data into small fragments (i.e., segments) and then sends each segment. If a user receives these segments of the data from multiple unspecific nodes, the user must verify the segments from the following two checkpoints:

(1) Is the received segment valid in itself? That is, the user must verify both the integrity of the segment and the authentication of the publisher of the segment.

It confirms the identity of the publisher who generated the segment and whether the publisher is valid. Additionally, it guarantees whether the segment has been unchanged during transmission. However, even if the validity of a segment is acceptable based on the first checkpoint, it is not sufficient because the verified segment may not be a piece of the data that the user requires. Hence, the second checkpoint is also necessary.

(2) Is the received segment a piece of the requested data?

An MHT-based authentication scheme (MHT-Auth) efficiently provides proper answers for the two checkpoints.

The MHT is a binary tree with $K = 2^k$ leaf nodes. Let N_i be a node with sequential index i. N_1 is the root node of the MHT. Each internal node N_i has two child nodes, N_{2i} and N_{2i+1}. Let V_i be the node value of N_i. Each V_i is computed as follows:

(1) If N_i ($2^k \leq i \leq 2^{k+1} - 1$) is a leaf node of the MHT, V_i is the hash value of a segment assigned to the leaf node N_i.
(2) If N_i ($1 \leq i \leq 2^k - 1$) is the internal node (or the root node) of the MHT, $V_i = h(V_{2i}||V_{2i+1})$, where $h(x)$ is a one-way cryptographic hash function.

Figure 1 shows that NDN constructs the MHT to verify data consisting of 16 ($K = 2^k = 2^4$) segments: $\{s[i] \mid 0 \leq i \leq 15\}$. A data publisher builds a binary tree with 16 leaf nodes and assigns the 16 segments to the leaf nodes of the binary tree according to the order of the segments. Since a segment $s[i]$ is assigned to the leaf node N_{K+i} of the MHT, V_{K+i} is $h(s[i])$. Then, starting from high-level nodes (i.e., from nodes in Level 3), it calculates all

node values in order. Afterward, the publisher generates the digital signature for the V_1 of the MHT, not for each segment $s[i]$. Then, it packages both a segment $s[i]$ and the digital signature. This package, denoted as $p[i]$, assumes an NDN's data packet type.

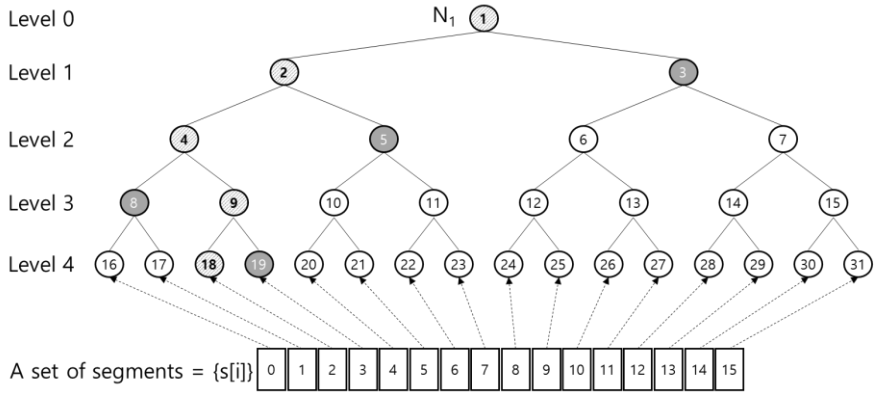


Figure 1. The structure of the MHT to verify 16 segments of data.

If a user receives the $p[i]$ of the data, to verify the $s[i]$ packaged in $p[i]$, it must compute V_1 of the MHT corresponding to the data. Therefore, it computes all node values of nodes on a full path, from N_{K+i} to N_1. The path is called as the *verification path* of $s[i]$. Given a node value V_r, if the user can obtain the sibling node value of N_r, the user can easily compute the parent node value of N_r. Hence, if the user obtains the sibling node values of the nodes on the path, it can compute all node values from V_{K+i} to V_1. For example, in Figure 1, when verifying $s[2]$, it can compute V_{18} using $s[2]$. Hence, if it obtains the V_{19} of N_{18}'s sibling node, it can compute V_9 of N_9 on the path.

The set of such sibling node values for all nodes on the path is called as a *witness* of a segment. For each segment $s[i]$, the data publisher generates a *witness* $w[i]$ and then packages the $w[i]$ into the $p[i]$. Thus, $p[i]$ consists of $s[i]$, $w[i]$, and the digital signature. If a user receives the $p[i]$, using both the $s[i]$ and the $w[i]$, it can easily compute V_1. As shown in Figure 1, if a user receives $p[2]$ for transmitting a segment $s[2]$, $w[2]$ consists of $\{V_{19}, V_8, V_5, V_3\}$. Hence, it can compute V_{18}, V_9, V_4, V_2, and V_1 as follows: $V_{18} = h(s[2])$, $V_9 = h(V_{18}||V_{19})$, $V_4 = h(V_8||V_9)$, $V_2 = h(V_4||V_5)$, and $V_1 = h(V_2||V_3)$.

A user receiving a segment $s[i]$ of data can check the integrity of $s[i]$, the publisher of, $s[i]$, and whether $s[i]$ is a piece of the data as follows:

(1) The integrity of the $s[i]$: NDN does not tell the hash value of the original $s[i]$ to the user. Therefore, even if the user computes the hash value of the received $s[i]$, it cannot confirm whether the calculated hash value is correct. However, NDN provides the digital signature for the valid V_1 of the MHT. If the received $s[i]$ is changed for some reason, so that the calculated hash value of the received $s[i]$ is different from the hash value of the original $s[i]$, the user must compute an incorrect V_1 and fail to verify the digital signature. Hence, the user can check the integrity of the received $s[i]$.

(2) The publisher of the $s[i]$: If the user checks the digital signature, the user can authenticate the publisher of the $s[i]$.

(3) Whether $s[i]$ is a piece of the requested data: If the received $s[i]$ is a segment of data other than the requested data, the user can check the verification of the computed V_1, even if $s[i]$ itself is valid. Since the MHT is uniquely built for each data point, the computed V_1 must be different from the V_1 of the MHT of the requested data. Hence, the user can confirm whether $s[i]$ is a correct piece of the requested data.

2.2. The Inefficiency of the MHT

2.2.1. The Iterated Processing of the MHT

When verifying data using an MHT-based authentication scheme, a user receiving the segments of the data must compute the same hash value several times. In this paper, such a

procedure, repeatedly calculating the same hash value, is described as *an iterated procedure*. Figure 2 shows the impact of iteration of the MHT. In Figure 2, 'x (y)' denotes the number of iterations to compute each node value. A receiver computes the same node value x times. Additionally, the same node value is transmitted y times as witnesses. For example, N_2 is labeled with x $(y) = 8(8)$. Here, $x = 8$ means that a receiver computes the same V_2 8 times when verifying 8 segments assigned to $\{N_{16}, \ldots, N_{23}\}$. Additionally, $y = 8$ means that the same V_2 is transmitted 8 times as witnesses for 8 segments assigned to $\{N_{24}, \ldots, N_{31}\}$.

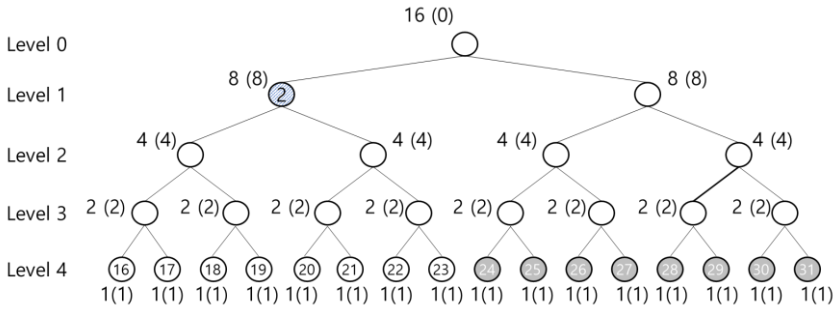


Figure 2. The inefficiency of the MHT-based authentication scheme caused by duplicated processing.

As shown in Figure 2, the number of iterated processes is dependent on the size of the MHT. If the MHT has $K = 2^k$ leaf nodes, a user verifying all segments of the data must compute the same V_2, 2^{k-1} times. For $K = 2^4$, the user computes the same V_2 only 8 times, but if $K = 2^{16}$, the user must calculate the V_2 a total of 32,768 times. Because of that, the user must calculate hash values 1,048,576 times to verify the data consisting of 2^{16} segments, even if the user needs only 136 hash values.

2.2.2. Impact Analysis of MHT Inefficiency

To describe the impact of the MHT's iterative process on service performance, Figure 3 shows the evaluation results of a digital signature-based segment authentication scheme and an MHT-based segment authentication scheme regarding computation time. Additionally, it shows the comparison of the computation time for verifying data with the download time for retrieving the data.

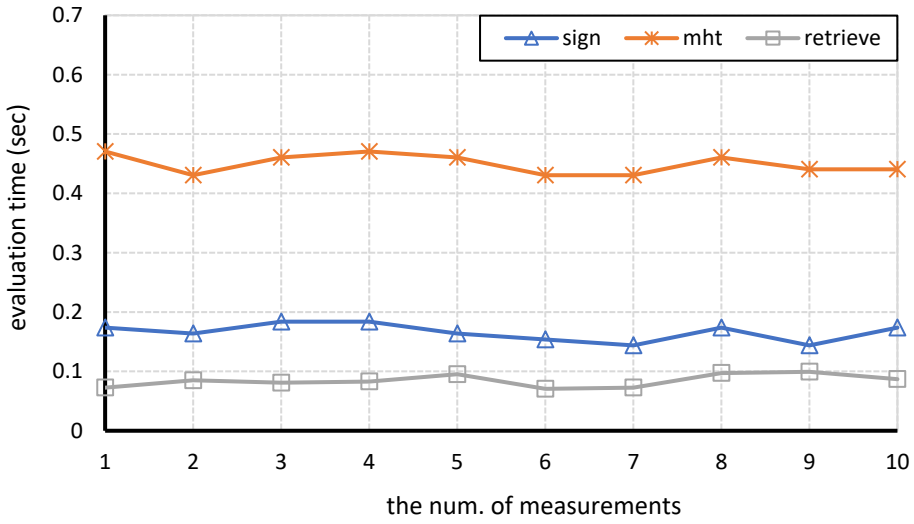


Figure 3. The performance evaluation of data authentication schemes.

The analysis uses flooding-based name routing and CCN implementation [6]. It takes the network size of 100 nodes as in [26]. It uses 1024 ($= 2^{10}$) segments, and each

segment packet size is 4 K bytes. In this case, the average end-to-end delay measurement for 100 segments is 8 ms.

Figure 3 demonstrates the measurement of the end-user service performance of a device (AMD Opteron 8354 2.2 GHz processor under Linux) 10 times in the following three cases.

(1) Case 1 ([retrieve]): The device measures the response time to retrieve all segments without verifying the segments. It shows 0.083 s on average.
(2) Case 2 ([sign]): It verifies the digital signature (RSA 2048) for each segment after receiving the segment. Then, it measures the total signature verification time.
(3) Case 3 ([mht]): It uses MHT consisting of 1024 leaf nodes and SHA 512. Measuring the performance of computing hash values does not verify the digital signature of the root node value of the MHT.

As shown in Figure 3, computing the hash values of the MHT is the main cause of service delay under the NDN environment. Especially, it shows that the latency in computing hash values takes longer than verifying the digital signature and receiving data segments. When considering the number of segments (2^n), if $n > 4$, the latency in verifying the digital signature of each segment is shorter than that of computing the node values of the MHT. Generally, it is known that the performance of a hash function is very efficient. However, severe inefficiency of MHT occurs when the total number of calls to a hash function is large, as in the NDN environment.

3. MHT Process Improvement

3.1. Segment Structure

In the MHT-based authentication scheme, a general data structure is as follows [27]:
Data = [Data-Type][Length][MetaInfo][Content][DataSignature]

We propose a new data segment structure for the case that does not require authentication by a data creator. The proposed structure is as follows:
Data = [Data-Type][Length][MetaInfo][Content][RootValue][DataSignature]

RootValue is the valid root node value of the MHT. The data publisher generates the digital signature for the root node value. We propose that the verifier optionally uses one of the two values, *RootValue* and *DataSignature*, as needed.

3.2. Improved MHT Verification

3.2.1. Verifying the Digital Signature in Improved MHT

It is a burden to verify the digital signature for each segment. Therefore, two methods to handle the digital signature verification of MHT are proposed. First, if a verifier needs to authenticate the data creator of a received segment, when the verifier retrieves $p[0]$, including the first segment $s[0]$ of data, it computes V_1, and verifies the digital signature in the $p[0]$. If the signature is valid, it saves the computed V_1. Afterward, if it receives other $p[i]$ of the data, it does not need to verify the digital signature in the $p[i]$ again. Instead, it is sufficient to compute V_1 for $p[i]$ and compare the computed V_1 with the saved V_1.

Second, if the verifier does not need to authenticate the data creator, it compares the computed V_1 with the *RootValue* of $p[i]$. If two values are the same, it regards $p[i]$ as being valid without verifying the *DataSignatue* of $p[i]$.

3.2.2. Verifying Node Values in Improved MHT

We propose the concept of *the shortest verification path* of a leaf node. To implement the shortest verification path of a leaf node, if a user verifies a segment, it saves valid node values of nodes on the full verification path of the corresponding leaf node and then reuses the saved valid node value to verify the next order segment.

Denote $P_{(n,1)} = \{a_n, a_{n-1}, \cdots, a_1\}$ as the path having length n of a binary tree, where a_n is a leaf node of the binary tree, a_1 is the root node is the binary tree, and a_i is a child

node of a_{i-1} for $i = n, n-1, \cdots, 2$. Let $SP_{(n,r)} = \{a_n, a_{n-1}, \cdots, a_r \mid 1 \leq r\}$ be the sub-path of $P_{(n,1)}$.

Definition 1. *Let the number of leaf nodes of MHT be $K = 2^k$. Assume that a user saves valid nodes for verifying the $(i-1)$th segment assigned to the ith leaf node of the MHT. The shortest verification path of the ith segment is the sub-path $SP_{(k,r)} = \{a_k, a_{k-1}, \cdots, a_r \mid 1 \leq r \leq k-1\}$ of the full verification path $P_{(k,1)} = \{a_k, a_{k-1}, \cdots, a_1\}$ of the ith segment satisfying the following two conditions:*

(Condition 1) The a_r of $SP_{(k,r)}$ is the same as one of the saved valid nodes.

(Condition 2) r is the largest index that satisfies Condition 1.

Figure 4 demonstrates the concept of valid node values and the shortest verification path of a segment. In Figure 4, $m[i]$ is the internal storage used to record the valid node value of a node having level i. If a user receives a segment and the segment is valid, it stores the node value V_r calculated during the verification process for the segment to $m[i]$, where $i = \lfloor log_2 r \rfloor$ is the level of node N_r. The operation process is as follows:

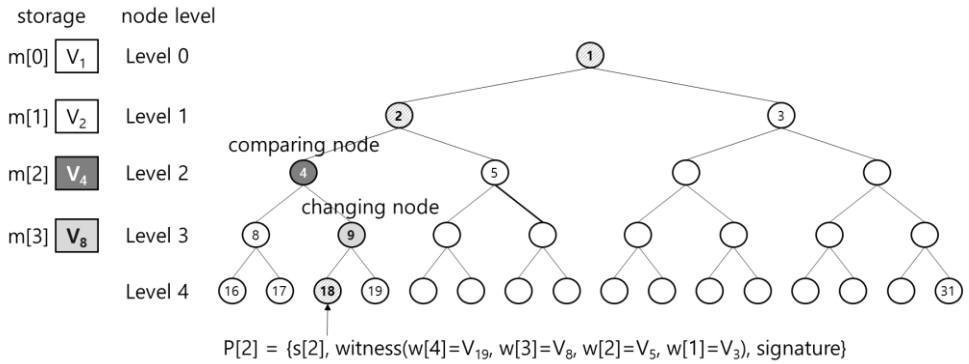


Figure 4. The segment verification process of the proposed scheme.

Step 1: A user receiving $p[0]$ verifies the $s[0]$ as the original MHT scheme does. If it needs to authenticate the $s[0]$ creator, it verifies the *DataSigature* of $p[0]$. Otherwise, it verifies the *RootValue* of $p[0]$. According to the characteristics of a secure hash function, given the *RootValue*, it is difficult to find a proper value such that the hash value of the found value is the *RootValue*. Even if an attacker changes both $s[0]$ and the *RootValue* to illegally pass this verification process, the attacker cannot create $s[1]$ to satisfies the same *RootValue*, so the attacker's attempt is eventually detected. Hence, verifying the *RootValue* is sufficient to check the integrity of $s[0]$. If $s[0]$ is valid, it saves the valid node values of the full verification path from the leaf node N_{16} of $s[0]$ to the root node N_1 according to their node levels.

Step 2: To verify the next segment $s[1]$, in principle, the user must compute all node values of nodes on the full verification path of $s[1]$ from leaf node N_{17} to N_1. To address such an inefficiency, we utilize *the shortest verification path* of $s[1]$, from the leaf node N_{17} to the internal node N_8. Therefore, the user computes only two node values, V_{17} and V_8, using witness data V_{16}. Then, the user compares the computed V_8 with the value of $m[3]$. The user has saved the valid node value V_8 in $m[3]$ when verifying $s[0]$. According to MHT characteristics, if the computed V_8 is the correct node value, then the node values of the ancestor nodes, i.e., V_4, V_2 and V_1, are always the correct node values. Therefore, if these two values are the same, it regards $s[1]$ as being valid. Hence, the user no longer needs to confirm the node values of the ancestor nodes.

Step 3: To verify the next segment $s[2]$, it is necessary to compute the node values of a shorter verification path from the leaf node N_{18} of $s[2]$ to N_4, not to N_1. After computing V_4, the user compares the computed V_4 with the value of $m[2]$. If these two values are the

same, it regards $s[2]$ as being valid. Now, it saves the computed V_9 to $m[3]$ for verifying the next segments.

Step 4: To verify the remaining segments, it repeats a process such as step 2 or 3.

Algorithm 1 describes the pseudocode for selecting a shorter verification path of a given segment, as well as verifying the segment. Let data consist of $K = 2^k$ segments. Hence, the MHT consists of $K = 2^k$ leaf nodes. As shown in Algorithm 1, a user requesting the data needs k internal storages $\{m[i] \mid i = 0, \ldots, k-1\}$ to save the computed valid node values. $\{w[i] \mid i = 1, \ldots, k-1\}$ is the witness of a segment, and $N[i]$ is the node value of a node having a level i.

Algorithm 1: Pseudo code for verifying a segment (Proposal 1).

Input data = {index, segment, witness = $\{w[k],\ w[k-1], \ldots,\ w[1]\}$, signature}
storage = $\{m[k-1], m[k-2],\ \ldots,\ m[0]\}$
Output result = {valid; invalid}, storage = $\{m[k-1], m[k-2],\ \ldots,\ m[0]\}$

```
Set mask:= 1
for level = k−1 down to 0 do
    if (index AND mask) is not zero then
        Stop this routine
    else
        Set mask:= mask *2
    end if
end for
Compute the hash value N[k] of the segment
for r = 0 up to level−1 do
    Read w[k−r]
    Compute N[k−r−1] using w[k−r] and m[k−r]
end for
if index is 0 then
    Verify the signature using N[0]
    if the signature is invalid then
        Return invalid
    end if
else
    if m[level] is not equal N[level] then
        Return invalid
    end if
end if
for r = 1 up to level−1 do
    Save N[r] in m[r]
end for
Return valid
```

The procedure of the proposed scheme is as follows:

(1) A user initializes every $m[i]$ with the NULL value.

(2) If the user receives $p[0]$ for the first segment $s[0]$ of the data, it computes all node values $\{N[k],\ N[k-1], \ldots,\ N[0]\}$ of nodes on a full verification path from the leaf node of $s[0]$ to N_1. These values are $\{V_K,\ V_{K/2},\ V_{K/4},\ \ldots,\ V_1\ \}$. Using the computed $N[0](= V_1)$, the user verifies the digital signature packaged in $p[0]$. If the signature is valid, it saves the computed node value $N[i]$ in $m[i]$ for $i = 0, \ldots, k-1$. That is, if a node N_r is on the path, the user saves the computed V_r to $m[i]$, where $i = \lfloor log_2 r \rfloor$.

(3) After that, the user receives $p[j]$ for the other segment $s[j]$ for $j > 0$. The user selects a full verification path of $s[j]$. It searches the node values of nodes on the verification path from the leaf node such that the node value $N[r]$ of the node on the path for $s[j]$ is equal to the value of $m[r]$. Let i be the first index found such that $N[i] = m[i]$. The index i is called the verification level of $s[j]$. A node having level i on the verification path of the $s[j]$ is called the *verification node* of the $s[j]$. A verification path from a leaf node of $s[j]$ to the verification node is called the *shortest verification path* of $s[j]$. Given a segment $s[j]$, the user

can easily find both the level and the shortest verification path of $s[j]$ as follows: The user finds the largest i such that $\left(j\ AND\ 2^{k-1-i}\right) \neq 0$, where AND is a bitwise AND operation. Then, the largest i is the level, and the shortest verification path of $s[j]$ is from the leaf node of $s[j]$ to the node having a lever i.

(4) Let the verification level of $s[j]$ be i. The user computes the node values $\{N[k],\ N[k-1], \ldots,\ N[i]\}$ of nodes on the shortest verification path of the $s[j]$.

(5) The user compares $N[i]$ with $m[i]$. If these two values are equal, the user regards segment $s[j]$ as valid.

(6) The user saves $\{N[k-1], \ldots,\ N[i-1]\}$ in $\{m[k-1], \ldots,\ m[i-1]\}$ according to their indices.

(7) The user repeats steps (3) through (6) until it finishes verifying the final segment of the data.

Moreover, if a segment having an even index i is valid, the user already has the valid hash value of the next segment having index $i+1$ because it is transmitted as a witness for $s[i]$. Hence, if the user adds an internal memory $m[k]$ and saves the witness to $m[k]$, then a user can utilize the value of $m[k]$ to verify the next segment $s[i+1]$. Algorithm 2 describes this process.

Algorithm 2: Pseudo code for verifying a segment (Proposal 2).

Input data = {index, segment, witness = $\{w[k],\ w[k-1], \ldots,\ w[1]\}$, signature}
storage =$\{m[k],\ \ m[k-1], m[k-2],\ \ldots,\ m[0]\}$
Output result = {valid; invalid}, storage =$\{m[k],\ \ m[k-1], m[k-2],\ \ldots,\ m[0]\}$

```
1: if index is odd then
2:    Compute the hash value of the segment
3:    Compare the hash value with m[k]
4:    if the two values are equal then
5:       Return valid
6:    end if
7: else
8:    Perform the pseudo code of Algorithm 1
9:    if segment is valid then
10:      Save w[k] to m[k]
11:      Return valid
12:   end if
13: end if
14: Return invalid;
```

(1) The process to verify a segment having an even index i is the same procedure described in Algorithm 1, except for additionally saving the hash value of the next segment $s[i+1]$ to $m[k]$.

(2) If a segment has an odd index i, the user computes the hash value of the segment and then compares the hash value with the value of $m[k]$ that was saved in the previous segment having an index $i-1$ verification process. If these two values are equal, the user regards the received segment as being valid.

4. Result

In this section, the computational overhead incurred for a user to calculate hash values is analyzed. It is assumed that the data have $K = 2^k$ segments.

In the original MHT operation, a user must compute $k+1$ hash values to verify each segment. Hence, the user computes $2^k(k+1)$ hash values to verify the data. Using the proposal 1 of Algorithm 1, the user computes each leaf node value once and each internal node value twice. Hence, the user computes 2^k hash values for leaf nodes and $2(\sum_{i=0}^{k-1} 2^i) = 2\left(2^k - 1\right)$ hash values for internal nodes. That is, the user computes $3 \times 2^k - 2$ hash values to verify the data. When using the proposal 2 of Algorithm 2, it reduces the

hash value computations by 2^{k-1}. That is, the user computes $5 \times 2^{k-1} - 2$ hash values to verify the data. In this case, the user must need k additional memory spaces.

Figure 5 shows the comparison results of the computational overhead for four cases: [proposal1] is the hash operation time of the proposal 1 of Algorithm 1, [proposal2] is the hash operation time of the proposal 1 of Algorithm 2, [mht] is the hash operation time of the original MHT, and [retrieve] is the data transmission time of NDN. The same configuration as Figure 3 was utilized. As shown in Figure 5, the performance of the proposed schemes was improved by more than three times. Practically, the greater the number of segments, the more obvious the improvement.

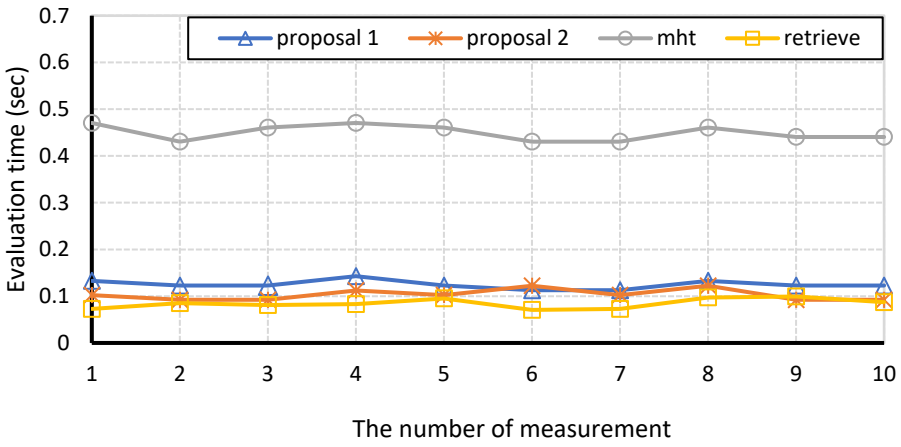


Figure 5. The performance comparisons of the proposed schemes with the computing time of the original MHT and with the downloading content time of NDN.

Figure 6 shows the evaluation comparison results of computational overheads for the four cases when transmitting 2^n segments, $10 \leq n \leq 19$. If $n = 10$, it means that a user retrieves 2^n (=1024) segments for [retrieve] seconds and verify the segments for [mht] (or [proposal 1] or [proposal 2]) seconds. Especially, as shown in Figure 6, if $n \geq 14$, verifying segments using MHT takes longer than verifying the segments using proposal 1 and 2 as well as receiving the segments. Practically, to transmit 1GB size content through NDN, the content publisher generates 262,144 segments and transmits the segments to a user. In this case, the size of MHT is $n = 14$. Compared to [mht], an increase in the number of segments has little effect on [proposals 1 and 2].

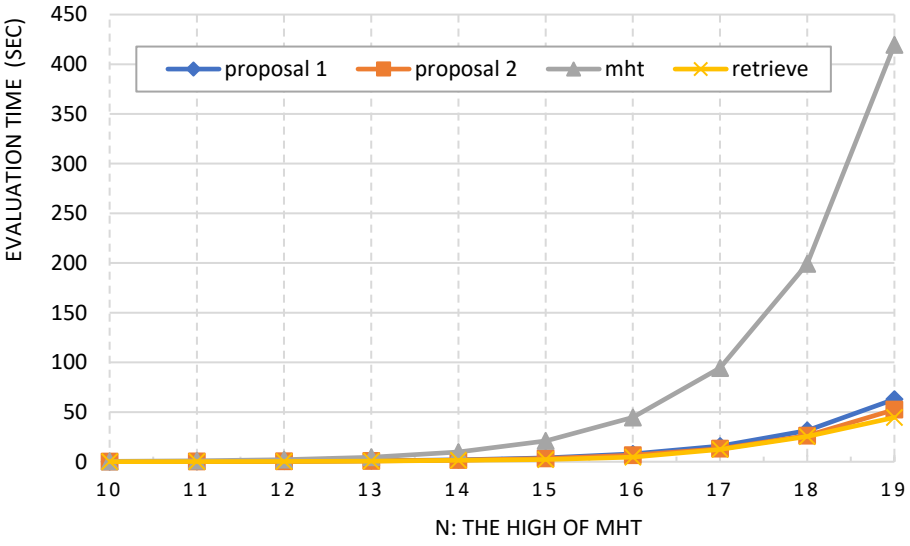


Figure 6. The performance evaluation of the proposed schemes for authenticating 2^n segments.

5. Conclusions

MHT has been utilized in various services to efficiently verify a data segment set. However, since the performance of MHT depends on the number of data segments, some applications/services requiring fast delivery of large amounts of content, such as multimedia services and connected vehicles, need to optimize MHT operations to enhance service performance. Therefore, the proposed scheme using saved node values and the shortest path was presented to reduce the computational overhead and service latency due to the MHT's iterated hash computation process.

This paper contains the following two contributions. First, MHT's inefficiency and the impact of such an inefficiency are demonstrated. A hash function is speedy and efficient, but even such an efficient function can lead to inefficiencies if it operates multiple times. The performance of computing hash values of MHT depends on the size of MHT, specifically on the number of MHT leaf nodes. In this paper, it is shown that in MHT, computing node values can cause more severe latency than verifying digital signatures and receiving segments.

Second, to improve the inefficiency of MHT, the concept of the shortest verification path of MHT, reusing node values that a verifier has proven to be valid previously, is proposed. For that task, an easy way to select the shortest verification path for each segment using the sequence number of the segment is demonstrated. Except for the first segment, not every segment needs to compute node values of nodes on the full path from a corresponding leaf node to the root node. Instead, it is sufficient to utilize a shorter verification path in which the length is between 2 and $k-1$ if the number of segments is 2^k. The shorter the length of a verification path is, the fewer the number of times to calculate the node value.

Future developments of this type of research include applying the proposed method to NDN-based applications/services such as connected vehicles and IoT-based home security services, and edge computing services. Such services require high capability for fast processing and fast transmission for multimedia data delivery. Generally, connected vehicles, smart home applications, IoT-based disaster management, and edge computing services are considering large-capacity IoT devices with ARM v7 (or v8) and 1–4 GB of memory or higher. Hence, the time-memory tradeoff of the proposed scheme is suitable for such applications/services.

Author Contributions: Conceptualization, D.K. and J.L.; methodology, D.K.; validation, D.K. and J.L.; formal analysis, D.K.; writing—original draft preparation, D.K.; writing—review and editing, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NRF of Korea, grant numbers NRF-2021R1F1A1062954 and NRF-2020R1F1A1070215.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ianni, M.; Masciari, E.; Sperlí, G. A survey of Big Data dimensions vs Social Networks analysis. *J. Intell. Inf. Syst.* **2020**, *57*, 73–100. [CrossRef] [PubMed]
2. Shen, H.; Li, Z.; Chen, K. Social-P2P: An Online Social Network Based P2P File Sharing System. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *26*, 2874–2889. [CrossRef]
3. Zhang, G.; Liu, W.; Hei, X.; Cheng, W. Unreeling Xunlei Kankan: Understanding Hybrid CDN-P2P Video-on-Demand Streaming. *IEEE Trans. Multimedia* **2014**, *17*, 229–242. [CrossRef]
4. Ahlgren, B.; Dannewitz, C.; Imbrenda, C.; Kutscher, D.; Ohlman, B. A survey of information-centric networking. *IEEE Commun. Mag.* **2012**, *50*, 26–36. [CrossRef]

5. Arshad, S.; Azam, M.A.; Rehmani, M.H.; Loo, J. Recent Advances in Information-Centric Networking-Based Internet of Things (ICN-IoT). *IEEE Internet Things J.* **2019**, *6*, 2128–2158. [CrossRef]
6. Jacobson, V.; Smetters, D.; Thornton, J.; Plass, M.; Briggs, N.; Braynard, R. Networking Named Content. In Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, Rome Italy, 1–4 December 2009; pp. 1–12.
7. Suwannasa, A.; Broadbent, M.; Mauthe, A. Impact of Content Popularity on Content Finding in NDN: Default NDN vs. Vicinity-based Enhanced NDN. In Proceedings of the 10th International Conference on Information Science and Technology (ICIST), Bath, London, Plymouth, UK, 9–15 September 2020; pp. 163–168.
8. Mehrabi, M.; You, D.; Latzko, V.; Salah, H.; Reisslein, M.; Fitzek, F.H.P. Device-Enhanced MEC: Multi-Access Edge Computing (MEC) Aided by End Device Computation and Caching: A Survey. *IEEE Access* **2019**, *7*, 166079–166108. [CrossRef]
9. Chen, Y.; Qiu, J.; Du, X.; Yin, L.; Tian, Z. Security of Mobile Multimedia Data: The Adversarial Examples for Spatio-temporal Data. *Comput. Netw.* **2020**, *181*, 107432. [CrossRef]
10. Sun, Y.; Li, M.; Su, S.; Tian, Z.; Shi, W.; Han, M. Secure Data Sharing Framework via Hierarchical Greedy Embedding in Darknets. *Mob. Netw. Appl.* **2019**, *26*, 940–948. [CrossRef]
11. Merkle, R.C. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology—CRYPTO '87*; LNCS; Springer: Berlin/Heidelberg, Germany, 1987; Volume 293, pp. 369–378.
12. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 1 September 2022).
13. Asif, M.; Aziz, Z.; Ahmad, M.; Khalid, A.; Waris, H.; Gilani, A. Blockchain-Based Authentication and Trust Management Mechanism for Smart Cities. *Sensors* **2022**, *22*, 2604. [CrossRef] [PubMed]
14. Patan, R.; Manikandan, R.; Parameshwaran, R.; Perumal, S.; Daneshmand, M.; Gandomi, A.H. Blockchain Security Using Merkle Hash Zero Correlation Distinguisher for the IoT in Smart Cities. *IEEE Internet Things J.* **2022**, *9*, 19296–19306. [CrossRef]
15. Prabhu, S.; Subramanyam, N.; Krishnan, S.; Sachidananda, B. Decentralized Digital Currency System using Merkle Hash Trees. 2022. Available online: https://arxiv.org/abs/2205.03259 (accessed on 1 September 2022).
16. Niaz, M.; Saake, G. Merkle Hash Tree based Techniques for Data Integrity of Outsourced Data. *Comput. Sci.* **2015**, *1366*, 66–71.
17. Mykletun, E.; Narasimha, M. Providing Authentication and Integrity in Outsourced Databases using Merkle Hash Tree's. In *UCI-SCONCE Technical Report*; 2003; Available online: http://people.eecs.berkeley.edu/~{}raluca/cs261-f15/readings/merkleodb.pdf (accessed on 1 September 2022).
18. Yang, N.; Chen, K.; Liu, Y. Towards Efficient NDN Framework for Connected Vehicle Applications. *IEEE Access* **2020**, *8*, 60850–60866. [CrossRef]
19. Papadopoulos, C.; Shannigrahi, S.; Afanaseyv, A. In-vehicle networking with NDN. In Proceedings of the 8th ACM Conference on Information-Centric Networking (ICN '21), Paris, France, 22–24 September 2021.
20. Ali, Z.; Shah, M.; Almogren, A.; Din, I.; Maple, C.; Khattak, H. Named Data Networking for Efficient IoT-based Disaster Management in a Smart Campus. *Sustainability* **2020**, *12*, 3088. [CrossRef]
21. Ahmed, S.H.; Kim, D. Named data networking-based smart home. *ICT Express* **2016**, *2*, 130–134. [CrossRef]
22. Yu, Y.; Li, Y.; Du, X.; Chen, R.; Yang, B. Content Protection in Named Data Networking: Challenges and Potential Solutions. *IEEE Commun. Mag.* **2018**, *56*, 82–87. [CrossRef]
23. Yu, Y.; Afanasyev, A.; Clark, D.; Claffy, K.; Jacobson, V.; Zhang, L. Schematizing Trust in Named Data Networking. In Proceedings of the ACM Conference on Information-Centric Networking (ICN), San Francisco, CA, USA, 30 September – 2 October 2015; pp. 177–186.
24. Yu, Y.; Afanasyev, A.; Seedorf, J.; Zhang, Z.; Zhang, L. NDN DeLorean: An Authentication System for Data Archives in Named Data Networking. In Proceedings of the 4th ACM Conference on Information-Centric Networking, Berlin, Germany, 26–28 September 2017; pp. 11–21.
25. Boussaha, R.; Challal, Y.; Bouabdallah, A.; Bessedik, M. Optimized in-network authentication against pollution attacks in software-defined-named data networking. *J. Inf. Secur. Appl.* **2019**, *50*, 102409. [CrossRef]
26. Liu, H.; Azhandeh, K.; Foy, X.; Gazda, R. A comparative study of name resolution and routing mechanisms in information-centric networks. *Digit. Commun. Netw.* **2019**, *5*, 69–75. [CrossRef]
27. NDN Packet Format Specification: Data Packet. 2022. Available online: https://named-data.net/doc/NDN-packet-spec/current/index.html (accessed on 1 September 2022).