The security of the BLT signature scheme with practical implementation issues

Trieu Quang Phong, Vo Tung Linh

***Abstract*—KSI infrastructure is a time-stamping and server-based signature solution deployed in Estonia to provide the integrity and timing information of data. With the goal of providing a personal signature that does not depend on the long-term secrecy of signing keys, Buldas et al. have proposed a BLT scheme that is based on the KSI infrastructure. Although Buldas et al. have shown that the (modified) BLT scheme is secure in the theoretical model, the fact that its practical implementation does not follow that description. In this paper, we will evaluate and provide two attack scenarios if the BLT schema is implemented in such a way, and then propose a solution to that problem.**

***Tóm tắt*—Hạ tầng KSI là một giải pháp chữ ký dựa trên máy chủ chứa thông tin thời gian được thử nghiệm triển khai ở Estonia để cung cấp tính toàn vẹn và thông tin về thời gian của dữ liệu.Với mục tiêu tạo ra một chữ ký số cá nhân mà không phụ thuộc vào tính bí mật lâu dài của các khóa ký, Buldas và các cộng sự đã đề xuất lược đồ BLT dựa trên nền của hạ tầng KSI. Mặc dù, Buldas và các cộng sự đã chỉ ra rằng lược đồ BLT (sửa đổi) là an toàn trong mô hình lý thuyết, nhưng triển khai thực tế của lược đồ này không hoàn toàn tuân theo mô tả đó. Trong bài báo này, chúng tôi sẽ đánh giá và đưa ra hai kịch bản tấn công nếu lược đồ BLT được triển khai như vậy, và sau đó đề xuất một giải pháp cho vấn đề đó.**

Keyworks—the BLT signature scheme, KSI infrastructure, non-repudiation, Merkle tree.

Từ khóa—Lược đồ chữ ký BLT, hạ tầng KSI, giả mạo tồn tại, tính chống chối bỏ, cây băm Merkle.

I. INTRODUCTION

In the current era of development of science and technology, *digital signature* (a digital version of handwritten signatures) has become a familiar concept. The main function of a digital signature is to provide authenticity and integrity to data.

Traditional digital signatures (for example, RSA, DSA, ECDSA, etc.) are implemented on the public key infrastructure (PKI). Their security is based on the long-term secrecy of the private key (for the signing function) and the assumption of the difficulty of the underlying problems (such as discrete logarithm or factorization problem). Therefore, there are two major problems with the use of traditional digital signatures:

- Revoking the private key when it expires or disclosure;
- Risk of attacks using quantum computations.

Key revocation for PKI signatures has been inherently an important issue and required a complex process (including the OCSP online certificate state protocol, TSA truth-time-stamping authority, and the listing certificate revocation of CRL). On the other hand, quantum-computational attacks will bring potential dangers in the future. Therefore, the authors A. Buldas, R. Laanoja, and A. Truu [5] proposed the BLT signature scheme with the following purposes:

- Signatures are compact in size and independently verifiable, and their verifiability does not require any communication with third parties;
- Signature generation keys can be instantly revoked without managing certificate revocation lists;
- Integrity of signatures does not depend on the long-term secrecy of keys, i.e. signatures stay verifiable after their generation. It implies that signatures can be reliably verified without assuming continued secrecy of some keys;

This manuscript is received on April 20, 2021. It is commented on April 29, 2021 and accepted on May 19, 2021 by the first reviewer. It is commented on June 02, 2021 and accepted on June 28, 2021 by the second reviewer.

- Trusted third parties are unable to forge clients' signatures, i.e. the signatures can be used for *non-repudiation*;
- Signatures are immune to quantum computational attacks.

Furthermore, this scheme is implemented on the KSI infrastructure [4], a globally distributed system for providing server-based signature and time-stamping services. More particularly, the BLT is a personalized version of the signature that the KSI infrastructure provides. The main idea of the BLT signature scheme is for the signer to commit a sequence of signing keys such that each key is designated to be used at a specific time. To sign a message m at some time t, a signing request is generated by hashing m and the signing key at that time by the client, and then is sent to the KSI service. The response received from the KSI service and the signing key at time t together becomes part of the signature on m. On the other hand, since the KSI infrastructure provides solid proofs of the time at which a document, assert, software, etc. was created or registered, there is a need for *synchronization between the used time of the signing key and the time-stamp that the KSI service provides.*

Theoretically, the security of BLT has been shown in the works [8], [9]. In particular, BLT provides *unforgeable* and *non-repudiation* properties.

However, the practical implementation of the BLT requires some modifications to ensure the soundness and stability in the signature generation process and thus does not maintain the original description of this scheme (which is secure theoretically). These modifications are to ensure synchronization between *the intended time of the signing key and the time-stamp received from the KSI service*. However, the practical implementation of the BLT requires some modifications to ensure the soundness and stability in the signature generation process and thus does not maintain the original description of this scheme (which is secure theoretically). These modifications are to ensure synchronization between the intended time of the signing key and the time-stamp received from the KSI service. More specifically, to sign a message m at time t, a client may send several requests in parallel by combining m with Δ signing keys at $t, \dots, t + \Delta$ correspondingly, where Δ is the maximum accepted service delay. Hence, there is always $t' \in \{t, \dots, t + \Delta\}$ for which t' is equal to the time in the response from the KSI service. However, we found that such a modification would make the BLT signature vulnerable to forgery and no longer guarantee non-repudiation.

Therefore, the purpose of this article is to examine some practical aspects of BLT implementations that may affect its theoretical security results, and then propose solutions to these problems.

II. Definitions

A. Merkle Tree

Hash tree was first introduced by Merkle ([1], [2]). This is a tree data structure using a "2-to-1" hash function $H: \{0,1\}^{2n} \to \{0,1\}^n$. Each node of the tree contains a hash value n-bits. Also, the node in the tree is either a *leaf* (with no child nodes) or an *internal node* (with two child nodes). The hash value x of an internal node is computed as $x \leftarrow H(x_l, x_r)$, where x_l and x_r are the hash values contained in the left and right child, respectively. There is one root node in the tree is not a child of any node. We will use $r \leftarrow \mathcal{T}^H(x_1, \dots, x_N)$ to denote a hash tree whose N leaves contain N the values $x_1, \dots, x_N$ and whose root node contains r.

Hash chain. This is a proof to show the participant of a value x_i in computation of the root hash r. It contains the values of all the siblings of the nodes on the unique path from x_i to the root in the tree. For example, to prove that x_3 belongs to the tree on the left in Fig. 1, we need to provide the values x_4 and $x_{1,2}$ which allow the verifier to compute $x_{3,4} \leftarrow h(x_3, x_4)$, $r \leftarrow h(x_{1,2}, x_{3,4})$, essentially recovering a slice of the tree, as shown on the right in Fig. 1. We will use $x \overset{c}{\rightsquigarrow} r$ to denote the computation r from x via the hash chain c.

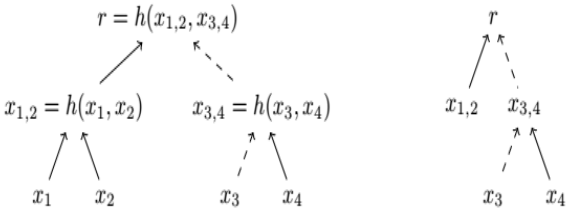

Fig. 1. The hash tree $\mathcal{T}^H(x_1, x_2, x_3, x_4)$ and the corresponding hashchain c for x_3.

B. KSI Infrastructure

The KSI infrastructure was originally developed in 2007 to include a signature token in any digital file to increase authentication efficiency. By combining the hash of a file along with the others on the requests received by the server in the same period into a Merkle hash tree, we will get the *keyless signatures* (or *time-stamps*) for them. In this infrastructure, the root hash values for each period are cryptographically linked together in a globally unique hash tree called a hash calendar. Hence, the *content* and *signing time* of a file is guaranteed that it is difficult to modify. This is an instance of blockchain.

The root hashes per period are stored in the database of the KSI infrastructure so that it is always available for signature verification. This storage grows to approximately 2 GB per year and scales over time, not by the number of requests.

To ensure scalability and handle a large number of requests, the KSI infrastructure is designed according to the architecture illustrated in Fig. 2:

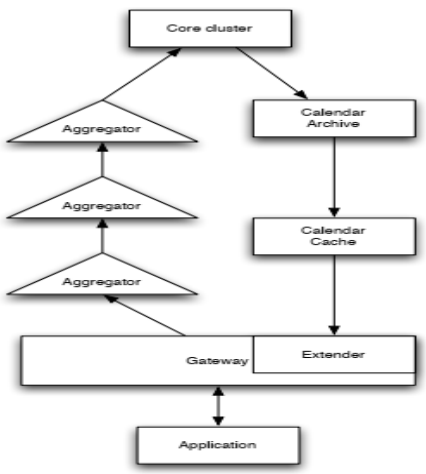

Fig. 2. High-level system architecture and aggregation network in the KSI infrastructure.

- A request is generated in the client application by hashing the document to be *signed (or time-stamped)*.
- The request is sent to *Gate* – a component of the system that provides the service to the end users. The gateway performs the initial aggregating of requests received during the same period into a Merkle tree, and then sends its root hash value to the higher server.
- The requests are then aggerated through the aggregators in the aggregation network.
- Finally, globally unique root hash value is computed by core cluster. Responses are sent back immediately through the aggregation network o answer the requests from clients.

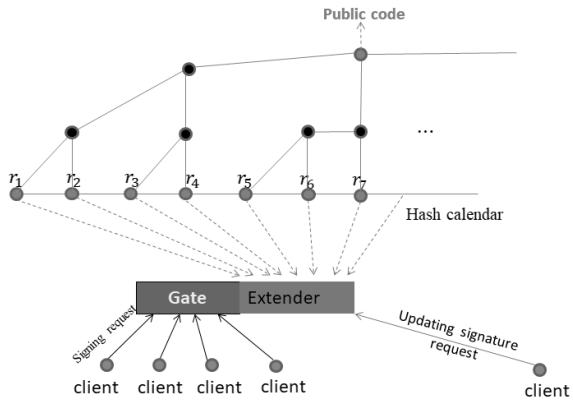

Fig. 3. The functionality of the extender service.

As mentioned, the root hash values of each period are stored in a calendar database. They are then linked together into a continuously growing hash tree whose root hash (named publication code) is periodically published. This database is distributed through the "calendar cache" layer to the extender service, usually colocated with the gateway server. The client applications use the extender service to update their signatures so that verifications may be done with the recent publication code. Note that this feature is not necessary for the traditional PKI signatures.

In [6], [7], the signatures provided by the KSI infrastructure are secure against back-

dating attacks. In other words, it is hard to modify the creation time of the signature to an earlier time, even with the support from the compromised servers.

C. The BLT signature scheme

The BLT signature scheme, which was first proposed by A. Buldas et al. [5], is a solution to move the keyless signature scheme in the KSI infrastructure into a personal signature. The main idea of the BLT signature scheme is that the signer commits a sequence of keys such that each key is intended for a specific time. Furthermore, after an intended key is used to sign a message, it will be released and involved in the signature.

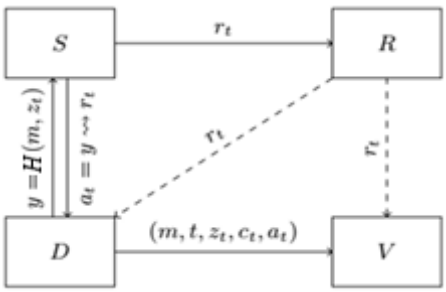

Fig. 4. Components of the BLT signature scheme.

In [8], [9], A. Buldas et al. presented several provable secure variants of the BLT signature scheme. This paper will use the description in [9], which is illustrated in Fig. 4 with the following participants:

- *Signer* who uses trusted functionality in secure device *D* to manage private keys.
- *Server S* (modeling the server system in the KSI service) that aggregates key usage events from multiple signers in fixed-length rounds and posts the summaries to append-only repository *R* (modeling the hash calendar).
- *Verifier V* who can verify signatures against the signer's public key p and the published value r_t obtained from *R*.

Key generation. To prepare to sign messages at times $t_0 + 1, \dots, t_0 + T$ (where $t = t_0 + 1$ is the time when the first key is intended to be used) the signer:

1. Generate T unpredictable n-bit signing keys $(z_1, \dots, z_T)$.
2. Compute T binding values $x_i \leftarrow H(i, z_i)$ for $i \in \{1, \dots, T\}$.
3. Compute the public key p by constructing a hash tree from the binding values: $p \leftarrow \mathcal{T}^H(x_1, \dots, x_T)$.

The aim of above tree data structure (Fig. 4) is to be able to extract the hash chains c_t linking the binding values to the public key: $H(i, z_i) \overset{c_i}{\rightsquigarrow} p$ for $i \in \{1, \dots, T\}$.

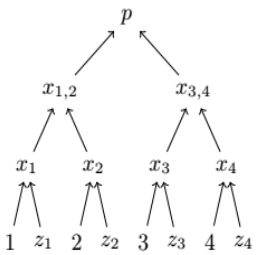

Fig. 5. Computation of public key for $T = 4$.

Signing. To sign message m at time $t = t_0 + i$, the signer:

1. Uses the appropriate key to commit a message: $y \leftarrow H(m, z_i)$.
2. Time-stamps the commitment y by sending it to S for aggregation and getting back the hash chain a_t linking y to the published value at time t: $y \overset{a_t}{\rightsquigarrow} r_t$.
3. Outputs the tuple (t, z_i, c_i, a_t).

Verification. To verify the message m and the signature $s = (t, z, a, c)$ with public key p and the published value r_t at time t, the verifier:

1. Checks that z was intended to use at time t: $H(i, z) \overset{c}{\rightsquigarrow} p$, where $i = t - t_0$.
2. Checks that m was committed with the key z at time t: $H(m, z) \overset{a_t}{\rightsquigarrow} r_t$.

Security issue [5]: The security of the signature scheme is based on the fact that if z_i is used right before $t_0 + i$ (when z_i expires), then it is impossible to abuse z_i. If z_i is used too early (sufficiently long before t), then z_i can be abused by anyone who has the signature with z_i. So, for the security of the scheme, it is viable that the signer verifies the signature

before disclosing it to other parties. This guarantees, due to the condition $t = t_0 + i$ that z_i is safe to disclose.

How to achieve $t = t_0 + i$ [5]: A BLT signature is valid only if $t = t_0 + i$, where t is the time indicated by the response from the service. However, this condition is hard to achieve for two following reasons: (1) There is a loss of the time synchronization between the client's signing device and the server, (2) even in the case of achieving the time synchronization, it is possible that $t \neq t_0 + i$ due to service latency which can result from several reasons related to system availability.

In the second case, the service can be organized so that the delay is predictable and is no more than a few seconds. Hence, the client may send several requests in parallel using the keys $z_i, z_{i+1}, \ldots, z_{i+\Delta}$, where Δ is the maximum accepted service delay. Therefore, we always obtain $i' \in [0, \ldots, \Delta]$ such that $t = t_0 + i'$. The client keeps the signature with such i' and deletes the rest.

III. Related results

A. The security model

Goldwasser et al. [3] provided a framework for analyzing the security of signature schemes where the attackers can access signing oracles by sending queries to achieve some requirements for the attack to be considered successful. In [3], the *existential unforgeability (EUF),* assuming that an attacker should be unable to forge signatures on any messages (even nonsensical ones), is considered the highest security requirement.

They also defined the *chosen-message attack* where the attacker can send some messages to be signed by the oracle and get back their signatures before returning a forged signature on a new message. In particular, as the one giving the attacker the most power, the *adaptive chosen-message attack* (ACM) in which the attacker will receive each signature immediately after submitting the message and can use any information gained from previous signatures to form subsequent messages.

Based on the above concepts, A. Buldas et al. [9] have restated *the existential unforgeability against adaptive chosen-message attack (EUF-ACM)* for their signature scheme.

Definition 1 [9]. *A signature scheme is S-secure existentially unforgeable against adaptive chosen-message attacks (EUF-ACM), if any T-time adversary, having access to a signer's public key p and to a signing oracle $\mathcal{S}$ to obtain signatures $s_1 \leftarrow SIGN(m_1), \ldots s_\ell \leftarrow SIGN(m_\ell)$ on adaptively chosen messages $m_1, \ldots, m_\ell$, an produce a new message-signature pair (m, s) such that $m \notin \{m_1, \ldots, m_\ell\}$, but s is a valid signature on m, with probability at most T/S.*

To formalize the security assumptions in Definition 1, A. Buldas et al. introduced three oracles:

- The first oracle is used to model the publishing of the root hashes of the time-stamping aggregation trees, and denoted as $\mathcal{R}$ (Fig. 6, right) that allows each r_t to be published just once.
- The signing oracle $SIGN$ (Fig. 6, left) will compute the requests at any time, but will only release the keys that have already expired for signing.
- The hash function H is modeled as a random oracle.

The signing oracle $SIGN$

Query $Sig(m, t)$:
  return $h(m, z_t)$
Query $Get(t, SIGN)$:
  If $i \geq t$ then:
    return (z_t, c_t)
  else:
    return $\perp$

The respository $\mathcal{R}$

Initialize:
  $i \leftarrow 0$
Query $Put(r)$:
  $i \leftarrow i + 1$
  $r_i \leftarrow r$
Query $Get(t, \mathcal{R})$:
  If $i \geq t$ then:
    return r_t
  else:
    return $\perp$

Fig. 6. The oracles in Definition 1.

The adversary $\mathcal{A}$ will be interacting with the oracles as shown in Fig. 6 to produce a forgery. We note that in this model, the service servers may be fully controlled by the adversary; only the repository $\mathcal{R}$ needs to be

trusted to operate correctly. This is to capture the cases in which the adversary has support from corrupted servers.

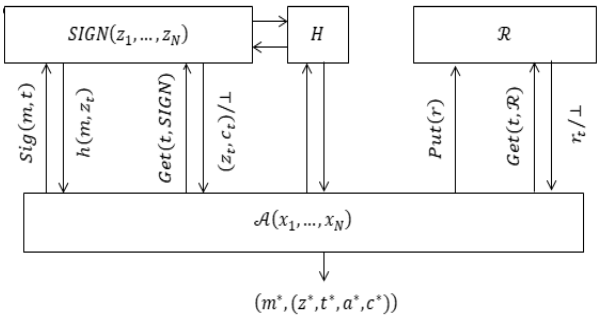

Fig. 7. The adversary's interactions with the oracles.

Normally, the signing process for a message m involves first calling $Sig(m,t)$, then committing to $\mathcal{R}$ the root hash value r_t of the aggregation tree at time t (via $Put(r_t)$ call), and then calling $Get(t,SIGN)$. Hence, the query from $\mathcal{A}$ is completely signed by $SIGN$ if $\mathcal{A}$ made two calls to $SIGN$ in the above order. In particular, A. Buldas et al. extended Definition 1 with the following condition.

Definition 2 [9]. *The pair (m,s) produced by an adversary is a successful forgery if s is a valid signature on m, but the adversary did not make the calls $Sig(m,t)$, $Get(t,SIGN)$ to the oracle $SIGN$, in that order, for any $t \in \{1,\dots,N\}$.*

B. The security of the BLT signature scheme

Below, we present the results on the security of the BLT signature scheme [9] according to Definition 1 (with the additional condition under Definition 2).

Theorem 3 [9]. *The BLT signature scheme, when instantiated with a hash function $H:\{0,1\}^{2n} \to \{0,1\}^n$ indistinguishable from a random oracle, is at least $2^{n/2-1}$- secure existentially unforgeable against adaptive chosen-message attacks by any T-time adversary.*

According to Definition 2, it is worth to note that if the adversary makes only one of the calls $Sig(m,t)$ or $Get(t,SIGN)$ to the signing oracle $SIGN$ and then produces s, such that s is a valid signature on m at time t, then the adversary is still considered successful. In particular, by using the call $Get(t,SIGN)$, the adversary can obtain the expired keys and the corresponding hash chains. Combined with Theorem 3 we have the assurance that it is hard to provide a valid BLT signature by using the expired keys.

IV. Analyzing the Security of the BLT Signature Scheme with Practical Implementation

The BLT signature scheme is implemented on top of KSI infrastructure and is proven to achieve EUF-CMA security. In Part III, we called this security outcome in a model where the adversary could have complete control of the service servers, except the public repository. Therefore, *even with support from the service servers*, it is hard to create a valid BLT on any message without knowing the signing keys. Furthermore, we also have the same results if the adversary only knows the expired signing keys. Therefore, once a valid BLT signature on some message has been generated at user $\hat{A}$'s request, then $\hat{A}$ cannot refuse it. In other words, BLT also provides *non-repudiation*.

As mentioned, one of the problems with the practical implementation of the BLT signature scheme is time synchronization. More specifically, a BLT signature is valid only if the intended time for the signing key is equal to the time provided in the time-stamp from the KSI service. A solution to that problem (mentioned by A. Buldas et al. in [5], [9]) was presented in Section C Part II.

However, we found that the BLT signature scheme is no longer secure theoretically when applying such a solution. In the following, we will give our analysis of this issue. These will relate to the non-repudiation of the BLT signature scheme.

Firstly, we return to the security of the BLT signature scheme, where Definition 1 requires that the adversary is successful if it can forge a valid signature for a "new" message (i.e., the message was never entirely signed before). However, in some cases, forging signatures for a message but at two different times also brings unexpected consequences. For example, we consider a scenario: *"Assuming that customer $\hat{A}$ has just created a signature to complete a business contract (such as, "$\hat{A}$ orders ten cars from $\hat{B}$. $\hat{A}$ will pay B after receiving the goods. In the case that $\hat{A}$ cancels the contract, he needs*

to pay $\hat{B}$ an amount equivalent to 50% of the contract value") at some point time-stamped by service. If $\hat{B}$ is able to forge $\hat{A}$'s signature on the same contract at a later time, $\hat{B}$ can ask $\hat{A}$ to be responsible for the contract which $\hat{A}$ did not agree". In the above case, the two mentioned contracts have the same content. However, because the time-stamps when they were confirmed are different, so there is a lot of trouble for customers. Therefore, it is important to prevent forging the signature on the signed messages at a later time. And Definition 2 provides such an addition to Definition 1. In other words, the original description of the BLT makes it achieve the above additional requirement. However, we will point out that such a requirement is no longer satisfied if the BLT is set up with the time synchronization solution discussed above.

Indeed, the adversary (in the role of verifier) can execute as follows:

- Firstly, when client $\hat{A}$ sends a sequence of requests $H(m, z_t), \dots, H\left(m, z_{t+\Delta_t}\right)$ to the KSI service, it is possible that the adversary interrupts these requests and repeatedly sends them to the service in next $\Delta - 1$ periods for getting back the time-stamps $a_{t+1}, \dots, a_{t+\Delta}$.
- In the second step, the adversary waits for $\hat{A}$ to produce a message and its signature for this period (assuming the labeled time is t') so that he knows the content of m.

In the final step, we consider two following scenarios:

Scenario 1: If $\hat{A}$ has sent another request to the KSI service for signing some message $m_1 \neq m$ and getting back its time-stamp at t_1 (such that $t' < t_1 < t + \Delta_t$), then the BLT signature for m_1 that $\hat{A}$ will produce is (z, t_1, a, c), where z is the intended key at the time t_1 and c is the hash chain linking the binding value of z to $\hat{A}$'s public key. In that case, $\hat{A}$ has used the key z two times instead of at most one time as described in [9]. Therefore, it is easy to see that the adversary can provide $\left(z, t_1, a_{t_1}, c\right)$ as a valid BLT signature of $\hat{A}$ on message m at time t_1. Indeed, it is based on the fact that a_{t_1} is the time-stamp that KSI infrastructure provided at the adversary's request $H(z, m)$ and z is a key of $\hat{A}$ that is intended to use at t_1. In summary, the attack scenario exploits two points:

(1) At the time t, the key z was used to creat a request for m before its intended time t_1; in this case z is still secrete because m was successfully signed with z' intended to used at time t' (where $t < t' < t_1$),
(2) The key z_{t_1} is released after getting back the time-stamp for m' from the KSI service.

This scenario would be appropriate in the case that $\hat{A}$ wants to generate signatures frequently.

Scenario 2: If $\hat{A}$ rarely generates signatures, then the above scenario is no longer suitable. However, we observe that it is still possible for the adversary to produce a valid forgery if a signing key is used to create several requests before its intended time. We note that such a key will be released only if its intended time is equal to the time in the response from the KSI service. In this case, we need an additional assumption that is acceptable for the BLT signature scheme, i.e the adversary is able to learn the expired keys. This stems from the following interesting observation. For a tree with four leaf values x_1, x_2, x_3, x_4, two internal node values $x_{12} = H(x_1, x_2)$, $x_{34} = H(x_3, x_4)$ and the root value $x_{1234} = H(x_{12}, x_{34})$, if we know the hash chain linking x_1 to the root value then we can also deduce the hash chain for x_2 and vice versa. It is similar to x_3 and x_4. Indeed, from the hash chain for x_1 is $\left((x_2, 0), (x_{34}, 0)\right)$, it is easy to obtain $\left((x_1, 1), (x_{34}, 0)\right)$ as the hash chain for x_2. Hence, the information of the hash chain for z_{t_0} is sufficient to find the hash chain for z_{t_0-1} or z_{t_0+1}. Therefore, if these two keys are both known and $t < t_0 < t + \Delta$, the adversary can extract a hash chain c and check whether it belongs to z_{t_0-1} or z_{t_0+1}. Without loss of generality, we assume that c is the hash chain for z_{t_0+1}, then the adversary can produce

$(z_{t_0+1}, t_0+1, a_{t_0+1}, c)$ as $\hat{A}$'s valid signature on m at t_0+1.

Thus, we have shown two possibilities for the adversary to find another signature on a signed message with a different confirmation time if the BLT signature scheme is implemented with the time synchronization solution proposed in [9]. Note that, according to the above scenarios, *the adversary can perform the attack without breaking any security properties of the hash*. The scenarios that we study are under the acceptable assumptions for the BLT signature scheme. In more detail:

- If a client creates the BLT signatures frequently, the former scenario would be appropriate to consider.
- On the other hand, if the client generates the BLT signatures infrequently, we need an additional assumption mentioned in the latter scenario. That assumption is acceptable because the security of the BLT signature scheme is guaranteed in [9] even if the adversary knows the expired keys. Furthermore, according to the original description of the BLT signature scheme in [4], once $\hat{A}$ produces his signature by using some signing key, the adversary will then find out all previous signing keys. Indeed, we recall that the signing keys in [4] are created by randomly generating a seed z_T and computing the sequence of keys $\{z_i\}_{i=1}^{T}$ via an one-way function f as $z_i = f(z_{i+1})$, for $i \in \{1, \dots, T-1\}$. Therefore, if a signing key z_i is used to sign some message successfully (i.e, it will become a part of the resulting signature), the all previous keys $z_1, \dots, z_{i-1}$ are known even if they have never been used before.

In order to prevent these two risks, we will present a solution based on the Merkle hash tree. However, we first explain why such risks may appear. The simple reason is that the adversary becomes aware of the "early signing requests" created by *using the keys before their intended time*, which is impossible with the original description of the BLT signature scheme. Therefore, our method is to make the adversary can't get the above advantage. In particular, in order to sign a message m at time t, $\hat{A}$ can execute as the following steps (assume that the first signing key is intended to be used at t_0+1):

- Generate Δ unpredictable nonce values $none_1, \dots, none_\Delta$, then compute Δ hash values $x_j = H\big(H\big(m, z_{i+j-1}\big), none_j\big)$ for $j \in \{1, \dots, \Delta\}$ and build Merkle tree $y \leftarrow \mathcal{T}^H(x_1, \dots, x_\Delta)$. We note that $i = t - t_0$.
- Send y as a signing request to the KSI service.
- Upon receipt of response a from the KSI service corresponding to time $t' \in [t, t+\Delta]$, $\hat{A}$ extract the hash chain c_H linking the hash value $H(H(m, z_{i'}), none_{t'})$ to y, and then assign $a := c_H \parallel a$ (for $i' = t' - t_0$). Finally, $\hat{A}$ retuns $(none_{i'}, z_{i'}, t', a, c)$ as a BLT signature on m, where c is the hash chain linking $H(i', z_{i'})$ to $\hat{A}$'s public key.
- We note that after producing the BLT signature on m, $\hat{A}$ has to delete Merkle tree $\mathcal{T}^H(x_1, \dots, x_\Delta)$, as well as the nonce values, in order to prevent the adversary from approaching them.

Now, in order to check the validity of a singnture $(none_t, z, t, a, c)$ on a message m with some public key p, the verifier can execute as follows:

- Check that c is the hash chain linking the binding value of z and $i = t - t_0$ to the public key p.
- Check that a linking $H(H(m, z), none_i)$ to $r_t \in \mathcal{R}$. Here, $\mathcal{R}$ is the hash calendar and r_t is the root hash value that the KSI infrastructure compute and put into $\mathcal{R}$ at time t.

With the above modification, to sign a message m the client only sends a single signing request y instead of sending several requests. Note that, given a hash tree, the knowledge of the root hash value y is not sufficient for the adversary to get information about any leaf nodes $H(H(m, z_{i+j-1}), none_j)$ (for $1 \le j \le \Delta$) whose values were used to build that tree.

Therefore, it is hard for the adversary to interrupt the "early signing requests" created by *using the keys before their intended time* and then exploit them in the same way as the two attacks in our scenarios.

Imagining that, if A creates a hash tree with leaf values $H(m, z_i), \dots, H(m, z_{i+\Delta})$ and then sends its root hash value as a signing request to the KSI service, it is also impossible for the adversary to get the leaf values back. So why do we need to use nonce values? This is because in case if the keys $z_i, \dots, z_{i+\Delta}$ are expired and exposed, then the adversary can completely recover $H(m, z_i)$,…, $H(m, z_{i+\Delta})$ from message m and the exposed keys. Hence, the use of nonce values in our approach is one solution that prevents the above problem even if the adversary knows the expired signing keys.

CONCLUSION

In this paper, we have considered the BLT signature scheme and its aspects of security and practical implementation. In [8] and [9], A. Buldas et al. have shown that BLT is EUF-CMA secure and non-repudiation. Furthermore, these properties are claimed even if the user's expired signing keys are exposed.

However, since BLT is a signature scheme based on the KSI infrastructure and the time-intended signing keys, a time synchronization solution has also been mentioned in [5], [9]. We have found the disadvantage of that solution, because it may break the non-repudiation of the BLT signature scheme in two scenarios where the attacker only needs to perform the interruption and replay operation of the "early signing requests" from the clients. In which, the former scenario will be suitable if the clients need to create signatures frequently. Otherwise, the latter one would be appropriate.

Finally, we have also proposed a solution based on a Merkle hash tree to eliminate the above disadvantages. Besides, we also want to analyze the security of the modified BLT version according to the model in [9]. And this is also our approach in the future.

REFERENCES

[1] R.C. Merkle. Secrecy, authentication, and public key systems. *Stanford university*, 1979.

[2] Merkle, R.C. Protocols for public-key cryptosystems. In: *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, pp. 122–134 (1980).

[3] S. Goldwasser, S. Micali, and R. L. Rivest. *A digital signature scheme secure against adaptive chosen-message attacks*. SIAM J. *Comput.*, 17(2):281-308, 1988.

[4] A. Buldas, A. Kroonmaa, R. Laanoja. *Keyless signatures' infrastructure: How to build global distributed hashtrees*. In: Nielson, H.R., Gollmann, D. (Eds.): NordSec 2013, LNCS 8208, pp. 313–320 (2013)

[5] Buldas, A., R. Laanoja, and A. Truu. *Efficient Quantum-Immune Keyless Signatures with Identity*. IACR Cryptol. ePrint Arch., 2014. 2014: p. 321.

[6] Buldas, A. and R. Laanoja. *Security proofs for hash tree time-stamping using hash functions with small output size*. in *Australasian Conference on Information Security and Privacy*. 2013. Springer.

[7] Buldas, A., et al. *Bounded pre-image awareness and the security of hash-tree keyless signatures*. in *International Conference on Provable Security*. 2014. Springer.

[8] Buldas, A., R. Laanoja, and A. Truu, *Security Proofs for the BLT Signature Scheme.* IACR Cryptol. ePrint Arch., 2014. 2014: p. 696.

[9] Buldas, A., R. Laanoja, and A. Truu. *A server-assisted hash-based signature scheme. in Nordic Conference on Secure IT Systems. 2017*. Springer.

[10] Trieu Quang Phong, Vo Tung Linh, Độ an toàn chứng minh được của lược đồ chữ ký Fiat-Shamir dựa trên ý tưởng của Pointcheval, *Journal of Science and Technology on Information Security*. Vol. 1, No. 1, 2015.

[11] Nguyen Tuan Anh, Trieu Quang Phong, Mối liên hệ giữa tính nhận biết tiền ảnh và một số tính chất mật mã khác của hàm băm, *Journal of Science and Technology on Information Security*. Vol. 13, No. 1, 2021.

ABOUT THE AUTHORS

Trieu Quang Phong

Email: phongtrieu53@gmail.com

Workplace: Institute of Cryptographic Science and Technology Government Information Security Committee

Eudcation: The BS in Department of Mathematics, Hanoi University of Science (2014).

Recent research direction: Provable security for the signature schemes and the key exchange protocol.

Vo Tung Linh

Email: votunglinh_mg@hus.edu.vn

Workplace: Institute of Cryptographic Science and Technology Government Information Security Committee

Eudcation: The BS in Department of Mathematics, Hanoi University of Science (2005); The MS in The BS in Department of Mathematics, Hanoi University of Science (2014).

Recent research direction: Signature schemes, Key exchange protocol, the public key cryptography.