A Blockchain-based Long-term Time-Stamping Scheme

Long Meng, Liqun Chen

Abstract. Traditional time-stamping services confirm the existence time of data items by using a time-stamping authority. In order to eliminate trust requirements on this authority, decentralized Blockchain-based Time-Stamping (BTS) services have been proposed. In these services, a hash digest of users' data is written into a blockchain transaction. The security of such services relies on the security of hash functions used to hash the data, and of the cryptographic algorithms used to build the blockchain. It is well-known that any single cryptographic algorithm has a limited lifespan due to the increasing computational power of attackers. This directly impacts the security of the BTS services from a long-term perspective. However, the topic of long-term security has not been discussed in the existing BTS proposals. In this paper, we propose the first formal definition and security model of a Blockchain-based Long-Term Time-Stamping (BLTTS) scheme. To develop a BLTTS scheme, we first consider an intuitive solution that directly combines the BTS services and a long-term secure blockchain, but we prove that this solution is vulnerable to attacks in the long term. With this insight, we propose the first BLTTS scheme supporting cryptographic algorithm renewal. We show that the security of our scheme over the long term is not limited by the lifespan of any underlying cryptographic algorithm, and we successfully implement the proposed scheme under existing BTS services.

Keywords: time-stamping, blockchain, long-term security

1 Introduction

Digital data has been widely adopted in the modern world. Time-stamping services are used to prove that a data item existed at a given point in time. For traditional centralized time-stamping services, a proof is created by a Time-Stamping Authority (TSA), who after receiving a data item from a user produces a verifiable cryptographic binding between the data and time, which is referred to as a time-stamp token [1, 2]. The security of this type of time-stamping services depends on both the security of the underlying cryptographic algorithms used to generate the token and the reliability and trustworthiness of TSAs.

In reality, TSAs may not always be reliable or trustworthy. If a TSA is compromised by an attacker, the validity of the time-stamp tokens from this TSA could be threatened no matter whether the underlying cryptographic algorithms are still secure or not. Therefore, the requirement on the reliability and trustworthiness of these central authorities is concerned as a weakness for traditional time-stamping services.

Since 2008, the innovation of the Bitcoin blockchain [3] has inspired people to explore more decentralized applications. Blockchain could be regarded as a public ledger, in which all committed transactions are stored in a chain of blocks [4]. A blockchain-based ledger has several advantages: (1) A blockchain is a decentralized system, so it minimizes the trust requirement on central authorities. (2) A blockchain is tamper-resistant, as transactions are validated by multiple decentralized nodes before being stored in a block. Once a block is validated and confirmed to be a part of a blockchain, any malicious modification of the transaction data in the block can be detected. (3) Each block contains a time-stamp when it is appended to the blockchain, so it is traceable that all the transactions in the blockchain exist at its corresponding block creation time.

Based on these advantages, several Blockchain-based Time-Stamping (BTS) services have been proposed [5–7]. In the "Proof of Existence" service [7], a web server collects a data item from a user, computes its hash value and embeds the result into a blockchain transaction. In the "OpenTimestamps" service [6] and "OriginStamp" service [8], a web server aggregates data items from users by using a Merkle tree [9], and inserts the tree root value into a blockchain transaction. The transaction record and the time-stamp in the block become the proof of existence of data items.

A BTS service makes use of hash functions and digital signature schemes to build a blockchain, and also uses hash functions to hash users' data. Obviously, the security of these services relies on the security of these underlying cryptographic algorithms. It is well-known that any hash function or signature scheme is only secure for a limited time period due to the operational life cycle or increasing computational powers of attackers [10]. Particularly, the upcoming quantum computers are considered to break most of the broadly-used signature algorithms and to increase the speed of attacking hash functions. However, for many types of digital data, such as identity information, health records, history archives etc, the proof of existence of data needs to be maintained for decades or even permanently, which is much longer than the lifetime of a single algorithm.

For the purpose of this work, if a scheme is secure in a long period of time that is not bounded with the lifetimes of its underlying cryptographic algorithms, we say that the scheme is *long-term secure*. If a BTS scheme is long-term secure, we refer to it as a *Blockchain-based Long-Term Time-Stamping* (BLTTS) scheme. Unfortunately, the topic of long-term security has not been addressed in the existing BTS services.

In this paper, we propose the first formal definition and security model of a BLTTS scheme. To construct such a scheme, we initially consider an intuitive solution that directly combines the existing BTS services and a long-term blockchain scheme [11], in which the hash functions and signature schemes used to build the blockchain could be securely transferred to stronger ones. But our proof shows that the solution is vulnerable to attacks after the hash function to hash user's data is compromised. In other words, the state-of-the-art solutions in this field show that a BLTTS scheme is still missing.

We fill this gap by proposing the first BLTTS scheme, which contains three solutions supporting the renewal of all underlying cryptographic algorithms. This is not a trivial target due to following challenges: 1) The cryptographic algorithms

are used both inside and outside the blockchain system, a comprehensive timeline to securely renew every algorithm is required. 2) Blockchain is a complex system that applies cryptographic algorithms in every block. 3) Each time-stamp renewal must be timely ordered with connections, since a verifier needs a complete time-stamping chain to prove the data existed before the earliest time-stamp. We formally prove that the security of our scheme is unbounded with the lifetime of any underlying cryptographic algorithm. Finally, we implement this scheme under the existing BTS services "OriginStamp" and "Opentimestamps", and the results show that our scheme is very efficient.

The remaining part of the paper is arranged as follows. We first review the related works in Section 2 and some preliminary building techniques in Section 3. In Section 4, we formally define the syntax and security model of a BLTTS scheme. Then we propose an intuitive BLTTS solution as an failure example and the first BLTTS scheme with three solutions as our main contribution in Section 5, and analyze the long-term security of the proposed scheme in Section 6. In the end, we implement the proposed scheme in Section 7 and conclude the paper in Section 8.

2 Related works

Traditional time-stamping. In 1990, Haber and Stornetta proposed the first prototype of digital time-stamping with two techniques: linear linking and random witness [12]. In 1993, Bayer et al. proposed a solution for time-stamp renewal [13]: the lifetime of a time-stamp could be extended by time-stamping the (data, time-stamp) pair with a new implementation before the old implementation is compromised.

In further years, the ideas of [12,13] have been adopted by multiple standards, especially the ISO/IEC standard [1,14,15] and ANSI standard [2]. Both standards specify time-stamping mechanisms and renewal mechanisms for long-term time-stamping services.

In addition, the ideas of [13] have been extended into several long-term integrity schemes [16–18], but the security analysis of such schemes were not given, until Geihs et al. formalized this idea separately into a signature-based long-term integrity scheme [19], and a hash-based long-term time-stamping scheme [20]. These two schemes provide substantial frameworks for analysing the security of long-term time-stamping schemes.

Recently, Long et al. proposed and analyzed a comprehensive long-term time-stamping scheme that allows the renewal of both client-side hash functions and server-side algorithms [21]. We are inspired from the ideas in [19,20], and [21] for our proposed schemes and security analysis.

Blockchain-based time-stamping. In 2008, Satoshi Nakamoto created the "Bitcoin" cryptocurrency system as the first blockchain prototype [3]. After that, dozens of blockchain-based cryptocurrencies were generated. For example, "Ethereum" was proposed as a developed blockchain platform that supports the creation of advanced smart contracts for achievable programs and commands [22]. During the past decade, there were many research surveys and reports on blockchain systems introducing their

structures, models, applications and challenges [4, 23–25]. In our paper, the structure of blockchain showing in Fig. 1 is learned from the remarked surveys and reports.

In 2015, the first BTS service "OriginStamp" was proposed [5]. Solutions similar to the OriginStamp are the "OpenTimestamps" project [6], and "Proof of Existence" service [7]. After that, there were many applications built on top of the "OriginStamp" service for various scenarios, which refer to manuscript submission [26], virtual patents [27], secure videos [28] etc. All of them leverage "OriginStamp" as a basis for time-stamping services. However, the long-term security of the OriginStamp, OpenTimestamps and Proof of Existence services has not been analyzed. The details of the existing BTS schemes are reviewed in Section 5.1.

Apart from the design of BTS services, some researches explored the reliability of the time-stamps included in the blockchain [29–33]. In these papers, the authors analyzed that the time-stamps in blockchains are not accurate and could be manipulated for attacks, and they proposed distinct solutions to this issue: [32], [30] and [29] had slightly different ideas with leveraging an external TSA since it can provide accurate time records; [33] claimed to integrate the hash value of a user's document with a constant number of latest confirmed blocks on the Ethereum blockchain; [31] proposed to use a smart contract that intermediates between a user and some time-stamp providers according to some selection strategy on the Ethereum blockhain. These ideas can be adopted for reliable and accurate blockchain time-stamps in our proposed scheme.

For the topic on how to insert data into a blockchain, Sward et al. provided a comprehensive survey for inserting arbitrary data into the Bitcoin blockchain [34]. Historical approaches were listed: Pay-to-Fake-Key-Hash (PF2KH), Pay-to-Fake-Public Key (PF2K), OP_RETURN, Pay-to-Fake-Multisig (P2FMS), Pay-to-Fake-Script-Hash (PFSH), Data Drop, and Data Hash Method. The authors made a comparison between these methods in terms of their efficiency, cost, scalability, and potential weaknesses. Besides, Gao et al. proposed a method to store data in the Bitcoin blockchain by encoding it into Bitcoin addresses [35], which enables more storage space for additional information of the data (e.g., file names, creator names, keywords). In our proposed scheme, the data insertion method can be selected based on these researches.

Long-term security of Blockchain. Regarding to the security of cryptographic algorithms in the blockchain, Giechaskiel et al. analyzed the impacts of broken cryptographic primitives on Bitcoin [36]. This work shows that the compromise of SHA-256, RIPEMD160 and ECDSA algorithms in the Bitcoin blockchain may cause the steal of coins, double spending, repudiated payments etc, in which any of them could be a devastating problem for Bitcoin security.

Following this work, Sato et al. proposed the first long-term blockchain (LTB) scheme with the renewal of hash functions and signatures used in a blockchain [37], and Chen et al. proposed an improved LTB scheme [38] to avoid the hard fork caused by the hash function renewal in [37] when using a proof-of-work blockchain.

Recently, Long et al. observed that [37, 38] only defined the transition from the first algorithm to the second one, and the security of those schemes are not analyzed. Then they proposed an enhanced LTB scheme [11] that enables algorithm renewal

in long-term periods, which has been proved secure under their proposed security model. The ideas of [11] are reviewed in Section 3.

3 Preliminaries

Blockchains. Blockchains are distributed digital ledgers of signed transactions that are grouped into blocks. A block is linked to its previous one by using hash functions after validation and undergoing a consensus decision [25]. In specific, each block is comprised of a block header and block data. As shown in Fig 1, a block header contains a block index number, a nonce, a hash value of the previous block header, a time-stamp, and a Merkle tree root hash value of all block data. The block data contains a list of transactions along with their corresponding digital signatures.

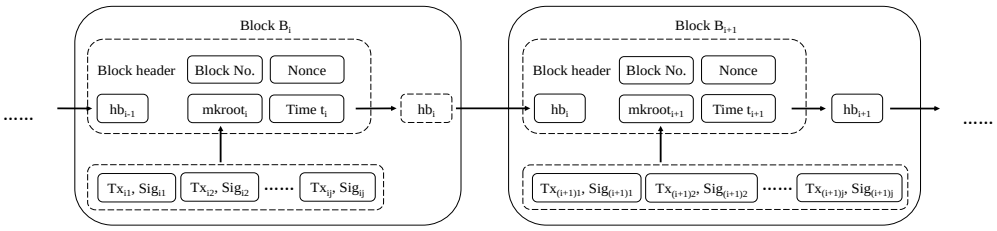

Fig. 1: The general structure of a blockchain

Blockchain technology utilizes cryptographic hash functions and signature schemes. In the block B_i in Fig. 1, each transaction is signed by the user who initiates the transaction, then all the transaction and signature pairs (Tx_{i1}, Sig_{i1}),...,(Tx_{ij}, Sig_{ij}) in the block are aggregated together by using a Merkle tree. The result root hash value $mkroot_i$ is stored in the block header for simplified verification [3]. The block header is then hashed into a hash value hb_i that is stored in the block header of the next block B_{i+1}. The signatures enable the network nodes to verify the integrity and authenticity of transactions, the chaining of hash values between blocks protects the integrity of block data.

Long-term Blockchain scheme. For a long-term blockchain, we review the ideas of the secure LTB scheme proposed by Long et al. [11], which could be divided into a hash transition procedure and a signature transition procedure.

The hash transition procedure (as shown in Fig. 2) is performed by the blockchain system. Assume at time $t_i (i \geq 1)$ when a hash function H_{i-1} becomes weak but not actually broken, the blockchain already has M blocks generated using hash function H_0, ..., H_{i-1} for calculating Merkle tree and block hash values. The transition from H_{i-1} to a stronger hash function H_i is performed with following steps: 1) divide all M blocks into r sets, with s blocks in each set, i.e., $M = r \times s$. 2) calculate an archive hash value of each set of blocks using H_i,

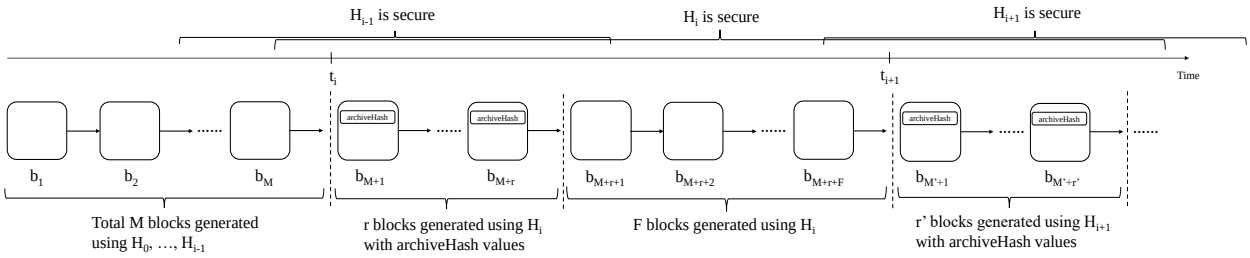

Fig. 2: The hash transition procedure of the LTB scheme proposed by Long et al.

i.e., $archiveHash_{i1} = H_i(b_1,\ ...,\ b_s)$, ..., $archiveHash_{ir} = H_i(b_{(r-1)s+1},\ ...,\ b_M)$, and stores $archiveHash_{i1}$, ..., $archiveHash_{ir}$ separately in the block header of $b_{M+1},\ ...,\ b_{M+r}$. Besides, $b_{M+1},\ ...,\ b_{M+r}$ are generated using H_i for calculating Merkle tree and block hash values. 3) The new blocks after b_{M+r} are generated using H_i and they do not include $archiveHash$ fields. Assume at time t_{i+1} when H_i becomes weak but still secure, there are total F blocks after b_{M+r}. Then set $M' = M + r + F$ and repeats steps 1-3: divide all M' blocks into r' sets, calculate archive hash values for each set using H_{i+1}, and store them into future blocks.

The Signature transition procedure is performed by users. Assume a user utilized a signature scheme $S_{i-1}(i \geq 1)$ for signing transactions in the blockchain. At the time when S_{i-1} is threatened but still secure, a new key pair should be generated from a stronger signature scheme S_i. Then the users' transactions should be transferred from the key pair of S_{i-1} to the new key pair of S_i, by using signature scheme S_{i-1}. i.e., $sig_i \leftarrow S_{i-1}(tx_i)$. The new transaction and signature pair (sig_i, tx_i) is then submitted to the blockchain. After that, users begin to sign new transactions using S_i.

4 Definitions of a BLTTS scheme

In this section, we provide the first formal definition and security model of a Blockchain-based Long-term Time-stamping (BLTTS) scheme.

4.1 Scheme definition

A BLTTS scheme includes following entities: a user, a blockchain system and a verifier. The user owns the data item to be time-stamped and sends it to the blockchain. The blockchain system stores the data in a block, which provides proof of existence of the data item. The verifier checks the validity of the time-stamp proofs.

Algorithms. A BLTTS scheme is comprised of a tuple of algorithms (BTSGen, BTSRen, BTSVer), which are defined as follows:

- $\mathrm{TS}_0 \leftarrow \mathrm{BTSGen}(C_0;\ \mathrm{D},\ blc)$: at time t_0, the time-stamp generation algorithm BTSGen takes input a data item D and a blockchain blc, outputs a time-stamp proof TS_0 by using a set of cryptographic algorithms C_0.

– $\mathrm{TS}_i \leftarrow \mathrm{BTSRen}(C_{i-1}, C_i; \mathrm{D}, blc)(i \in [1, n])$: at time $t_i (i \in [1, n])$ when any of the cryptographic algorithm in C_{i-1} is threatened but still secure, the time-stamp renewal algorithm BTSRen takes input a data item D and the blockchain blc, outputs a time-stamp proof TS_i by using a set of cryptographic algorithms C_i.
– $b \leftarrow \mathrm{BTSVer}(\mathrm{D}, \mathrm{TS}_0, ..., \mathrm{TS}_n, blc, \mathrm{VD}, t_v)$: at verification time t_v, the time-stamp verification algorithm BTSVer takes input a data item D, a group of time-stamp proofs $\mathrm{TS}_0, ..., \mathrm{TS}_n$, the blockchain blc, the verification data VD (defined in the further paragraph), and the verification time t_v, then outputs a bit $b=1$ if the time-stamp proofs are valid on D. Otherwise outputs $b=0$.

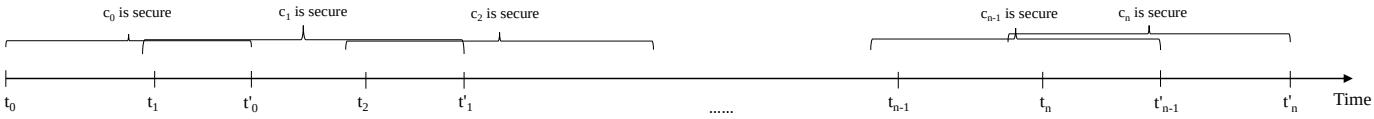

Fig. 3: Timeline of cryptographic algorithm lifetime and renewal

Timeline. Fig. 3 shows the relations between the lifetime and renewal time of every particular type of cryptographic algorithm $c_i \in C_i$. For $i \in [1, n]$, c_{i-1} should be renewed to a stronger one c_i when it becomes weak but still within its lifetime. In other words, at time t_i, c_{i-1} and c_i are secure. We denote the starting usage time and breakage time of c_i separately as $c.t_i$ and $c.t'_i$. For $C.t_i$ and $C.t'_i$, we mean the common starting usage time of all $c_i \in C_i$ and the breakage time of any $c_i \in C_i$.

Verification data. VD contains necessary data used for the BTSVer algorithm. Especially, VD must contain the information indicating the start time and breakage time of every $c_i \in C_i$ for $i \in [1, n]$. This information can be collected from reliable sources such as the NIST standard [39, 40]. Then at the time of verifying the validity of algorithms, the block time-stamps and the VD time should be synchronized with a same criteria, e.g., the global time.

4.2 Security model

In a BLTTS scheme, we make following assumptions:

1. The verification data VD is trusted.
2. Every time a hash function or signature scheme is threatened but still secure, a stronger hash function or signature scheme is available.

A BLTTS scheme should satisfy two properties: correctness and long-term integrity. The definitions of these two properties are given as follows:

Correctness. Correctness means that if all entities perform their functions correctly, a BLTTS scheme is able to prove the existence of data items in long-term periods that are not bounded with the lifetimes of underlying cryptographic algorithms.

Definition 1. *(Correctness.) Let* BLTTS = (BTSGen, BTSRen, BTSVer) *be a BLTTS scheme. For the scheme to be correct, it must satisfy that if time-stamp proofs* TS_0, ..., TS_n *are generated for any data item* D *by following the BTSGen and BTSRen algorithms, at time* $t_v \in [C.t_n, C.t'_n]$*, the verification algorithm outputs* $\mathrm{BTSVer}(\mathrm{D}, \mathrm{TS}_0, ..., \mathrm{TS}_n, blc, \mathrm{VD}, t_v) = 1$.

Long-term Integrity. The long-term integrity measures the probability of an attacker successfully compromising a BLTTS scheme. Intuitively, we say that an attacker is able to compromise a BLTTS scheme, if it is able to claim that a data item exists at a point in time but actually it does not exist, or to tamper with existing time-stamp proofs without being detected. Thereby, we say that a BLTTS scheme has long-term integrity if any polynomial time adversary is unable to compromise the BLTTS scheme in long-term periods that are not bounded with the lifetimes of underlying cryptographic algorithms.

To formalize this, the long-term integrity model is defined as a game running between a long-lived adversary $\mathcal{A}$ and a simulator $\mathcal{B}$. $\mathcal{B}$ has computational resources comparable to $\mathcal{A}$. $\mathcal{A}$ is able to access a clock oracle $clk(\cdot)$ and a blockchain oracle $Blc(\cdot)$, which are defined as follows:

1. $clk(\cdot)$: $P \leftarrow clk(t)$. $\mathcal{A}$ inputs a time point t to the oracle, the oracle returns the corresponding computational power P according to the timeline introduced in Section 4.1. That means, P develops with the increase of t but restricted within each time period. The ability that $\mathcal{A}$ can break or cannot break any algorithm depends on P.
2. $Blc(\cdot)$: $\mathrm{TS} \leftarrow Blc(x)$, $R \leftarrow R \| (x, \mathrm{TS})$. $\mathcal{A}$ inputs a data item x, the oracle submits x to the blockchain blc, and returns a time-stamp proof TS by following the BTSGen or BTSRen algorithm. At the same time, the oracle records x along with TS in a list R.

The long-term integrity experiment is displayed as Algorithm 1.

Algorithm 1: Long-term integrity (LTI) experiment $\mathbf{Exp}^{\mathrm{LTI}}_{\mathrm{BLTTS}}(\mathcal{A})$

```
Input: n, blc, VD
Output: a bit 1 or 0
Set R = [ ];
(x', TS_0, ..., TS_n) ← A^{clk(·), Blc(·)} /* R is updated for Blc(·) queries. */
if BTSVer(x', TS_0, ..., TS_n, blc, VD, t_v) = 1 and ∃(x', TS_0, ..., TS_n) ∉ R. then
    Return 1;
else
    Return 0;
```

We use $\mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1]$ to denote the probability of $\mathcal{A}$ winning the game in Algorithm 1. By the time t_v, we denote the probability that $\mathcal{B}$ breaks at least one

hash function within its validity period as $\mathcal{B}_{\mathcal{H}}^{Com}$, and the probability that $\mathcal{B}$ breaks at least one signature scheme within its validity period as $\mathcal{B}_{\mathcal{S}}^{Com}$.

Definition 2. *(Long-term Integrity.) Let* BLTTS = (BTSGen, BTSRen, BTSVer) *be a BLTTS scheme, let $\mathcal{A}$ and $\mathcal{B}$ be an adversary and a simulator respectively as specified above. Then the BLTTS scheme holds the long-term integrity property if for any point in time t_v, there exists a constant c such that* $\boldsymbol{Pr}[\boldsymbol{Exp}_{\mathrm{BLTTS}}^{\mathrm{LTI}}(\mathcal{A})=1] \leq c\cdot(\mathcal{B}_{\mathcal{H}}^{Com}+\mathcal{B}_{\mathcal{S}}^{Com})$.

5 The proposed BLTTS scheme

In this section, we first briefly show that why the existing BTS schemes do not satisfy the security requirement of a BLTTS scheme. Then we propose an intuitive BLTTS solution that directly combines the existing BTS schemes and the LTB scheme reviewed in Section 3, and prove that the solution does not hold the long-term integrity property of a BLTTS scheme. Thereafter, we propose the first successful BLTTS scheme, which is comprised of three solutions dependent on how the client-side data is processed before being written into a blockchain. Finally, we compare the advantages and drawbacks of each solution. The notation follows that in Table. 1.

$n \in \mathcal{N}$	Total number of cryptographic algorithm	D	Data item to be time-stamped
$i \in \{0, n\}$	Index number of cryptographic algorithm	C_i	i-th tuple of cryptographic algorithms
c_i	a particular type cryptographic algorithm in C_i	$c.t_i$, $c.t_i'$	The starting usage time and breakage time of c_i
cH_i	i-th Client-side hash function	sH_i, S_i	i-th server-side hash function and signature scheme
TS_i	Time-stamp proof generated using C_i	blc	The blockchain used for time-stamping scheme
t_v	The verification time of time-stamp proofs	h_i	Hash value computed through cH_i
b_i	The block provides the time-stamp proof TS_i	tx_i	The transaction stores the data item to be time-stamped
b_{prei}	The previous block of b_i	hb_i	Hash value of block b_i
bid_i	Index number of block b_i	sig_i	The digital signature of tx_i
$mkroot_i$	Merkle tree root value of block data in b_i	ts_i	Time-stamp included in block b_i
VD	The verification data used in BTSVer algorithm	pc, ps	Client and server-side hash path used in MT algorithm
$a \Leftarrow b$	Store parameter b into a	$a \subseteq b$	Parameter a is included in parameter b
$root \leftarrow \mathrm{MT}(H; \mathrm{D}, p)$	Merkle tree algorithm that takes input D with a hash path p using hash function H, outputs a root value $root$		

Table 1: Notation

5.1 Existing BTS schemes

The existing Blockchain-based Time-Stamping (BTS) schemes "Proof of existence" [7], "OpenTimestamps" [6], and "OriginStamp" [5] can be summarized as the black fonts in Fig. 4.

Since the existing BTS schemes do not specify the BTSRen algorithm, they do not comply with our BLTTS definition in Section 4.1. It is trivial to prove that the schemes are vulnerable to attacks after any of cH_0, sH_0 or S_0 is compromised.

5.2 Intuitive BLTTS solution

As reviewed in Section 3, the existing LTB scheme [11] supports the secure transition of server-side algorithms sH_0 and S_0. Intuitively, the guarantee of a long-term secure blockchain in the BTS schemes may be able to achieve a BLTTS scheme. Thus, we add a BTSRen algorithm and corresponding procedures in BTSVer algorithm in the existing BTS schemes by leveraging the LTB scheme (as the red fonts in Fig. 4).

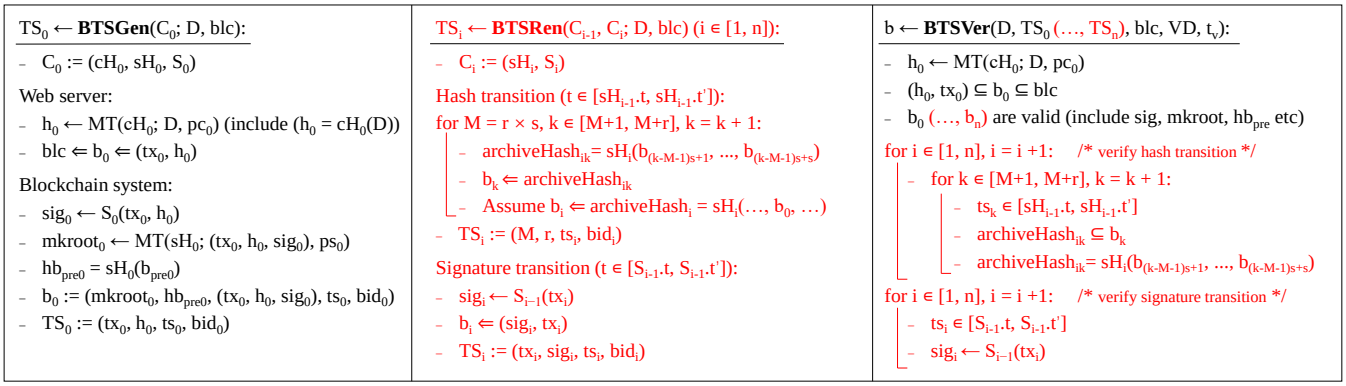

Fig. 4: An Intuitive BLTTS solution that directly combines the existing BTS schemes (black fonts) and a LTB scheme (red fonts)

Now we analyze the long-term security of the intuitive solution based on our security model proposed in Section 4.2.

Theorem 1. *The intuitive BLTTS solution specified in Fig. 4 does not hold long-term integrity property.*

Proof. At time $t \in [cH.t_0,\ cH.t'_0]$, $\mathcal{A}$ can firstly submit a hash value of data item x calculated using cH_0 to the oracle $Blc(\cdot)$, i.e., $h_0 = cH_0(x)$. The oracle returns TS_0 and records $(x,\ \mathrm{TS}_0)$ in the list R. After that hash function sH_0 and signature scheme S_0 could be transferred to stronger ones before they are compromised. For x, the hash transition can be described as: $sH_1(tx_0,\ cH_0(x))$, the signature transition can be written as: $S_0(tx_0,\ cH_0(x))$. But after cH_0 is compromised $(t_v > cH.t'_0)$, $\mathcal{A}$ is able to output $(x',\ \mathrm{TS}_0)$ with $sH_1(tx_0,\ cH_0(x)) = sH_1(tx_0,\ cH_0(x'))$ or $S_0(tx_0,\ cH_0(x)) = S_0(tx_0,\ cH_0(x'))$ that achieves $\mathrm{BTSVer}(x',\ \mathrm{TS}_0,\ blc,\ \mathrm{VD},\ tv) = 1$ and $(x',\ \mathrm{TS}_0) \notin R$ with non-negligible probability. Thus, Theorem 1 follows. □

Discussions. If a client-side hash function is used, a BLTTS scheme has two layers of security: the client-side hash function and server-side algorithms. For the BTS schemes, the algorithms at both sides are not renewed to stronger ones, so the adversary could attack any side after the algorithms are compromised. For the intuitive solution, despite the server-side algorithms can be transferred to stronger ones, the client-side

could be attacked. The reason is, the data item is not exposed to the blockchain after it is hashed. The long-term security at the server-side cannot guarantee the long-term security at client-side. So far, a BLTTS scheme does not exist. This motivates us to propose a BLTTS scheme (in Section 5.3) that satisfies long-term integrity.

5.3 Proposed BLTTS scheme with three solutions

Our proposed BLTTS scheme is composed of three different solutions as displayed in Fig. 5: 1) raw data is time-stamped without using client-side hash functions, 2) the hash value of data is time-stamped, and 3) the hash value of data together with the last time-stamp proof is time-stamped. Some parts of the algorithms are referred to Fig. 4.

Remarks. In our scheme, the user plays a similar role to the web server in the existing BTS services. Note that the security of the web server is not required to be assumed, because the web server only performs a hash calculation or a Merkle tree aggregation. Both calculations could be easily verified by the user. Thus, we collectively call the web server and user as a general "user", who is behalf of the data item holder.

Our scheme supports both client-side and server-side algorithm renewal. Thus, a renewed time-stamp proof TS_1, ..., TS_n could be either for client-side or server-side renewal. The difference is, the relations between server-side renewal proofs are explicitly recorded on the blockchain, so these proofs are not necessary to be obtained by users. On the contrary, the client-side renewal proofs are randomly distributed in blockchain transactions, users need to collect their proofs as related evidences for verification.

The time-stamps in the blockchain should be reliable and accurate to verify the start and breakage time of cryptographic algorithms. The solutions could be referred to related works [29–33, 41] in terms of detailed scenarios.

The method to insert a data item, a hash value or a hash value along with a time-stamp proof into a blockchain transaction depends on 1) which blockchain is selected for the BLTTS scheme, and 2) the specific size of the inputs. For instance, if a user has a small input (lower than 80 bytes) to submit on Bitcoin, OP_RETURN is the most efficient choice; for medium amounts of data (between 80 and 800 bytes), P2FMS is the most cost-effective option; for large amounts of data (beyond 800 bytes), the Data Drop w/o method provides the least expensive option [34]. The user should select a data insertion method that has enough capacity for the data item and while it is cost-effective.

5.4 Solutions Comparison

As Table 2 shows, we provide a comparison between Solution 1, 2, and 3 in terms of following factors (as the titles of Table 2 from left to right): 1) the renewal type that the user needs to perform, 2) whether the time-stamped data is exposed to public, 3) whether the data size is limited in each transaction, 4) whether the solution is cost-free, 5) whether there are connections between time-stamp proofs, and 6) the compatibility with existing BTS services. Then we analyze the best application scenario for each solution.

In Solution 1, a user directly submits the data item to the blockchain. The only action required for the user is to renew server-side signature schemes. Time-stamp

<table>
<tr><td colspan="3">Solution 1: Time-stamp the raw data</td></tr>
<tr>
<td>$TS_0 \leftarrow$ BTSGen(C_0; D, blc):
- $C_0 := (sH_0, S_0)$
User:
- $blc \Leftarrow b_0 \Leftarrow (tx_0, D)$
Blockchain system:
Change every h_0 to D in BTSGen of Solution 2
- $TS_0 := (tx_0, D, ts_0, bid_0)$</td>
<td>$TS_i \leftarrow$ BTSRen(C_{i-1}, C_i; D, blc) ($i \in [1, n]$):
- $C_i := (sH_i, S_i)$
Same as the BTSRen algorithm in Fig. 4
- Hash transition: $TS_i := (M, r, ts_i, bid_i)$
- Signature transition: $TS_i := (tx_i, sig_i, ts_i, bid_i)$</td>
<td>$b \leftarrow$ BTSVer(D, TS_0, …, TS_n, blc, VD, t_v):
- $(tx_0, D) \subseteq b_0 \subseteq blc$
- b_0, …, b_n (sig, mkroot, hb_{pre}) are valid
Add the BTSVer procedures on Fig. 4 for verifying hash and signature transitions</td>
</tr>
<tr><td colspan="3">Solution 2: Time-stamp the hash value of the data</td></tr>
<tr>
<td>$TS_0 \leftarrow$ BTSGen(C_0; D, blc):
- $C_0 := (cH_0, sH_0, S_0)$
User:
- $h_0 \leftarrow MT(cH_0; D, pc_0)$ (include $(h_0 = cH_0(D))$
- $blc \Leftarrow b_0 \Leftarrow (tx_0, h_0)$
Blockchain system:
- $sig_0 \leftarrow S_0(tx_0, h_0)$
- $mkroot_0 \leftarrow MT(sH_0; (tx_0, h_0, sig_0), ps_0)$
- $hb_{pre0} = sH_0(b_{pre0})$
- $b_0 := (mkroot_0, hb_{pre0}, (tx_0, h_0, sig_0), ts_0, bid_0)$
- $TS_0 := (tx_0, h_0, ts_0, bid_0)$</td>
<td>$TS_i \leftarrow$ BTSRen(C_{i-1}, C_i; D, blc) ($i \in [1, n]$):
- $C_i := (cH_i, sH_i, S_i)$
Client-side hash renewal ($t \in [cH_{i-1}.t, cH_{i-1}.t']$):
User:
- $h_i \leftarrow MT(cH_i; D, pc_i)$ (include $(h_i = cH_i(D))$
- $blc \Leftarrow b_i \Leftarrow (tx_i, h_i)$
Blockchain system:
Change every index from 0 to i in BTSGen
- $TS_i := (tx_i, h_i, ts_i, bid_i)$
Add the BTSRen algorithm in Fig. 4
- Hash transition: $TS_i := (M, r, ts_i, bid_i)$
- Signature transition: $TS_i := (tx_i, sig_i, ts_i, bid_i)$</td>
<td>$b \leftarrow$ BTSVer(D, TS_0, …, TS_n, blc, VD, t_v):
for $i \in [1, n]$, $i = i+1$: /* verify client-side hash renewal */
- $ts_i \in [cH_{i-1}.t, cH_{i-1}.t']$
- $h_i \leftarrow MT(cH_i; D, pc_i)$
- $(tx_i, h_i) \subseteq b_i \subseteq blc$
- b_i (sig_i, $mkroot_i$, hb_{prei}) is valid
Add the BTSVer procedures on Fig. 4 for verifying hash and signature transitions</td>
</tr>
<tr><td colspan="3">Solution 3: Time-stamp the hash value of the data and the previous proof</td></tr>
<tr>
<td>$TS_0 \leftarrow$ BTSGen(C_0; D, blc):
- $C_0 := (cH_0, sH_0, S_0)$
User:
- $h_0 \leftarrow MT(cH_0; D, pc_0)$ (include $(h_0 = cH_0(D))$
- $blc \Leftarrow b_0 \Leftarrow (tx_0, h_0)$
Blockchain system:
Same as the BTSGen in Solution 2
- $TS_0 := (tx_0, h_0, ts_0, bid_0)$</td>
<td>$TS_i \leftarrow$ BTSRen(C_{i-1}, C_i; D, blc) ($i \in [1, n]$):
- $C_i := (cH_i, sH_i, S_i)$
Client-side hash renewal ($t \in [cH_{i-1}.t, cH_{i-1}.t']$):
User:
- $h_i \leftarrow MT(cH_i; D, pc_i)$
- $blc \Leftarrow b_i \Leftarrow (tx_i, h_i \| TS_{i-1})$
Blockchain system:
Change h_i to $h_i \| TS_{i-1}$ in BTSRen of Solution 2
- $TS_i := (tx_i, h_i \| TS_{i-1}, ts_i, bid_i)$
Add the BTSRen algorithm in Fig. 4
- Hash transition: $TS_i := (M, r, ts_i, bid_i)$
- Signature transition: $TS_i := (tx_i, sig_i, ts_i, bid_i)$</td>
<td>$b \leftarrow$ BTSVer(D, TS_0, …, TS_n, blc, VD, t_v):
for $i \in [1, n]$, $i = i+1$:
- $ts_i \in [cH_{i-1}.t, cH_{i-1}.t']$
- $h_i \leftarrow MT(cH_i; D, pc_i)$
- $(tx_i, h_i \| TS_{i-1}) \subseteq b_i \subseteq blc$
- b_i (sig_i, $mkroot_i$, hb_{prei}) is valid
Add the BTSVer procedures on Fig. 4 for verifying hash and signature transitions</td>
</tr>
</table>

Fig. 5: Proposed BLTTS scheme with three solutions

proofs generated from server-side hash and signature transitions can be both collected from the blockchain with connections, so the user does not have to hold any time-stamp proof for verification. Since the data is not hashed and compressed, it is publicly readable and the data size is limited in each transaction. The existing BTS services only allow the insertion of hash value of the data item into a blockchain transaction, thus this solution is not compatible with the services. A user needs to

Solutions	Cryptographic renewal performed by users	Data exposure	Data size limit per transaction	Costs	Time-stamp connection	Compatibility with existing BTS services
1	Server-side signature scheme	Exposed	Limited	Not free	Both sides are connected	Not compatible
2	Client-side hash function and server-side signature scheme	Not exposed	Unlimited	Can be free	Only server-side is connected	Compatible
3	Client-side hash function and server-side signature scheme	Not exposed	Unlimited	Not free	Both sides are connected	Not compatible

Table 2: Comparison between Solution 1, 2 and 3 with multiple factors

insert data individually with a minimum non-dust amount of money for validating a transaction if the blockchain is used for cryptocurrency.

In Solution 2, a user submits a hash value of data item(s) to the blockchain. The user needs to renew both client-side hash functions and server-side signature schemes. Time-stamp proofs for server-side renewal are connected, but time-stamp proofs from client-side are just hash values without connections. The user needs to collect all time-stamp proofs for client-side hash renewal for verification. The data item is not exposed and the data size is unlimited because it is hashed, and it is the only form that the existing BTS services accept. Especially, Opentimestamps and OriginStamp provide free time-stamping services.

In Solution 3, a user submits a hash value of data item(s) with a previous time-stamp proof to the blockchain, which brings connections for time-stamp proofs generated from client side. The user only provides the last client-side time-stamp proof for verification. Besides, both client-side hash functions and server-side signature schemes are renewed by the user. Since the data item is hashed, it preserves data nondisclosure and unlimited data size. But the nested time-stamp proofs in TS_{i-1} will be harder to be inserted when the size becomes much bigger. This form of submission is not accepted by the existing BTS services, thus it also requires self-insertion by the user with a minimum non-dust amount of money for each transaction.

In summary, if data's privacy is not a primary goal to be considered, and the size of data is small enough to be inserted, Solution 1 is the perfect choice for users due to its convenience; if the nondisclosure of data is critical to be protected, or the data size is large, or the user cares most about the cost, Solution 2 is the best choice that can be implemented by the existing free BTS services; if data's nondisclosure and size matters, but the existence of data is required to be proved for a very long time, such as hundreds of years. It may be hard to keep every time-stamp proof for verification, then Solution 3 is a good option because it provides connections between time-stamp proofs.

6 Security analysis

We now prove that the proposed BLTTS scheme holding each security property in terms of the security models and definitions in Section 4.2.

6.1 Proof of correctness

Theorem 2. *The proposed BLTTS scheme holds the correctness property.*

Proof. In terms of the definition of correctness, we assume that a group of time-stamp proofs TS_0, ..., TS_n of a data item D are generated through algorithm BTSGen and BTSRen legitimately. At time $t_v \in [C.t_n, C.t'_n]$, the algorithm BTSVer takes input D, TS_0, ..., TS_n, VD, *blc* and t_v, and the verifications cover three parts: 1) the correctness of client-side renewal, 2) the connections between data item, transaction, block and the blockchain, and 3) the correctness of server-side renewal. We now analyze the output of BTSVer:

For Solution 1, by using algorithm BTSGen, the data item D is submitted to a block transaction tx_0 on block b_0 from blockchain *blc*. Then the client-side renewal is not required, and the connections between D, tx_0, b_0 and *blc* is guaranteed. By using algorithm BTSRen, the hash transition and signature transition can be both implemented before the previous server-side hash function sH_{i-1} or signature scheme S_{i-1} is compromised, thus the BTSVer algorithm outputs 1 and Solution 1 is correct.

For Solution 2 and 3, by using algorithm BTSGen, a hash representation h_0 of D is calculated by cH_0 and submitted to tx_0 on block b_0 from *blc*. Then if the algorithm BTSRen performs correctly, a new hash representation $h_i (i \geq 1)$ of D is calculated by using a stronger hash function cH_i before the previous one cH_{i-1} is compromised, and h_i (or $h_i \| \mathrm{TS}_{i-1}$) is submitted to tx_i on block b_i from *blc*. Thus, the client-side renewal of both solutions are correct, the connections between h_0, tx_0, b_0 and *blc*, and the connections between h_i (or $h_i \| \mathrm{TS}_{i-1}$), tx_i, b_i and *blc* are guaranteed. Same as Solution 1, the server-side hash transition and signature transition can be both implemented at the correct time by algorithm BTSRen, thus the BTSVer algorithm outputs 1 and Solution 2 and 3 are correct, then the theorem follows. □

6.2 Proof of long-term integrity

Theorem 3. *Assume the verification data VD is trusted, and every time a hash function or signature scheme is threatened but still secure, a stronger hash function or signature scheme is used for renewal respectively, then the proposed BLTTS scheme holds long-term integrity property.*

As the experiment defined in Section 4.2, the adversary $\mathcal{A}$ is able to input data item (or hash representation) to the blockchain oracle $Blc(\cdot)$ for obtaining time-stamp proofs. Thus, the long-term integrity of the scheme addresses the long-term security of server-side algorithms, and of the client-side hash functions. That means $\mathcal{A}$ can win the game through following two cases:

- Case 1: $\mathcal{A}$ correctly computes the hash representations of data items aligning with the VD archive, but wins the game by outputting a valid time-stamp, which was not through the blockchain oracle $Blc(\cdot)$.
- Case 2: $\mathcal{A}$ correctly queries the blockchain oracle $Blc(\cdot)$, but wins the game by outputting a valid time-stamp, which was not aligned with the VD archive.

We use $\mathbf{Pr}[\mathbf{Exp}^{\text{LTI, C1}}_{\text{BLTTS}}(\mathcal{A})=1]$ and $\mathbf{Pr}[\mathbf{Exp}^{\text{LTI, C2}}_{\text{BLTTS}}(\mathcal{A})=1]$ to denote the probability of $\mathcal{A}$ winning the game through Case 1 and Case 2 respectively. We use $\mathcal{B}^{Com}_{cH}$, $\mathcal{B}^{Com}_{sH}$, and $\mathcal{B}^{Com}_{S}$ to denote the probability that $\mathcal{B}$ breaks at least one client-side hash function, at least one server-side hash function, and at least one server-side signature scheme within their validity periods respectively. Then we prove Theorem 3 from Lemma 1 and Lemma 2 corresponding to Case 1 and Case 2.

Lemma 1. *There exists a constant c such that* $\boldsymbol{Pr}[\boldsymbol{Exp}^{\text{LTI, C1}}_{\text{BLTTS}}(\mathcal{A})=1] \leq c\cdot(\mathcal{B}^{Com}_{sH}+\mathcal{B}^{Com}_{S})$.

Proof. Since we adopt the existing LTB scheme [11] for server-side algorithm renewal, their proofs show that the LTB scheme satisfies the following two properties:

- Long-term integrity: there is negligible probability that $\mathcal{A}$ is able to claim a non-existed data item or to tamper data in any existing blocks on the blockchain without being detected in long-term periods.
- Long-term unforgeability: there is negligible probability that $\mathcal{A}$ is able to outputs a message m along with a valid signature s on m, and m was not previously signed by S on the blockchain in long-term periods.

More accurately, the proof of [11] reduces the probability that a polynomial-time adversary $\mathcal{A}$ wins the game through tampering any block data or forging any signature on the blockchain to the probability that $\mathcal{B}$ breaks at least a server-side hash function or signature scheme within its validity period, which is negligible. Thus, $\mathbf{Pr}[\mathbf{Exp}^{\text{LTI, C1}}_{\text{BLTTS}}(\mathcal{A})=1] \leq c\cdot(\mathcal{B}^{Com}_{sH}+\mathcal{B}^{Com}_{S})$ holds, and Lemma 1 follows. Besides, it directly leads to Theorem. 3 holding for Solution 1 in the BLTTS scheme since only server-side algorithms are used in the solution.

Lemma 2. *There exists a constant c such that* $\boldsymbol{Pr}[\boldsymbol{Exp}^{\text{LTI, C2}}_{\text{BLTTS}}(\mathcal{A})=1] \leq c\cdot(\mathcal{B}^{Com}_{cH})$.

Proof. In Case 2, $\mathcal{A}$ wins the game by outputting time-stamp proofs TS_0, ..., TS_n on a distinct data item $x' \neq x$, so that $\text{BTSVer}(x', \text{TS}_0, ..., \text{TS}_n, \text{VD}, blc, t_v)=1$. Besides, at time t_i for $i \in [1, n]$, the two corresponding client-side hash function cH_{i-1} and cH_i used by $\mathcal{A}$ must be both collision resistant. Now let us check the following reasoning:

At time t_0, $\mathcal{A}$ computes a hash representation $\text{MT}(cH_0; x, pc_0)$ of a data item x (pc_0 is empty for the case of a single hash computation of D), and obtains a time-stamp proof TS_0 from the blockchain oracle $Blc(\cdot)$. Assume hash function cH_0 is collision resistant at t_0.

At time t_1, $\mathcal{A}$ decides to renew the time-stamp proof TS_0 by using a stronger hash function cH_1. Since hash functions cH_0 is still collision resistant at this time, $\mathcal{A}$ can compute either $\text{MT}(cH_1; x, pc_1)$ and obtain a new time-stamp proof TS_1 (Case a), or $\mathcal{A}$ computes $\text{MT}(cH_1; x', pc'_1)$ and obtain TS_1 (Case b) from the oracle $Blc(\cdot)$. If $\mathcal{A}$ wins the game after Case b happens, it must hold that $\text{MT}(cH_0; x, pc_0) = \text{MT}(cH_0; x', pc'_0)$. Correspondingly, $\mathcal{B}$ can obtain the pair $((x, pc_0), (x', pc'_0))$ to break the collision resistance of cH_0 within its validity period. This result is contradict to the assumption that cH_0 is collision resistant at t_1. If Case a happens, let us carry on with our reasoning. We now assume that cH_1 is collision resistant at time t_1.

At time t_2, cH_0 may have been broken, but we assume that cH_1 is still collision resistant, and the hash representation $\mathrm{MT}(cH_1;\, x,\, pc_1)$ is a part of TS_1. Now repeating the previous situation, $\mathcal{A}$ can compute either $\mathrm{MT}(cH_2;\, x,\, pc_2)$ and obtains TS_2 (Case a), or determine $\mathrm{MT}(cH_2;\, x',\, pc'_2)$ and obtain TS_2 (Case b) from the oracle $Blc(\cdot)$. Again, if $\mathcal{A}$ wins the game after Case b happens, it must hold that $\mathrm{MT}(cH_1;\, x,\, pc_1) = \mathrm{MT}(cH_1;\, x',\, pc'_1)$. Correspondingly, $\mathcal{B}$ can obtain the pair $((x,\, pc_1),\, (x',\, pc'_1))$ to break the collision resistance of cH_1 within its validity period, which contradicts the assumption, and Case a leads us to continue our reasoning.

Carrying on our argument as before, only Case a for each time-stamp proof renewal is considered. We assume that cH_{n-1} is collision resistant at both t_{n-1} and t_n, and the hash representation $\mathrm{MT}(cH_{n-1};\, x,\, pc_{n-1})$ is a part of TS_{n-1}. If $\mathcal{A}$ finally wins the game after computes $\mathrm{MT}(cH_n;\, x',\, pc'_n)$ and obtains TS_n from the oracle $Blc(\cdot)$, $\mathrm{MT}(cH_{n-1};\, x,\, pc_{n-1}) = \mathrm{MT}(cH_{n-1};\, x',\, pc'_{n-1})$ must hold. Then $\mathcal{B}$ can obtain the pair $((x,\, pc_{n-1}),\, (x',\, pc'_{n-1}))$ to break the collision resistance of cH_{n-1} within its validity period.

In summary, based on the above reasoning, the probability that $\mathcal{A}$ wins the game through Case 2 is reduced to the same level of the probability that $\mathcal{B}$ breaks at least one client-side hash function within its validity period. Thus, $\mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI,\ C2}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] \leq c \cdot (\mathcal{B}^{Com}_{cH})$ holds, and Lemma 2 follows. □

Combining Lemma 1 and Lemma 2, the winning probability of $\mathcal{A}$ from both Case 1 and Case 2 is reduced to the same level of the probability that $\mathcal{B}$ breaks at least one client-side hash function, or at least one server-side hash function, or at least one server-side signature scheme within its validity period. There exists a constant c such that:

$$\begin{aligned} \mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] &= \mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI,\ C1}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] + \mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI,\ C2}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] \\ &\leq\ c \cdot (\mathcal{B}^{Com}_{cH} + \mathcal{B}^{Com}_{sH} + \mathcal{B}^{Com}_{\mathcal{S}}) \end{aligned} \tag{1}$$

With aggregating $\mathcal{B}^{Com}_{cH}$ and $\mathcal{B}^{Com}_{sH}$, we have:

$$\mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] \ \leq\ c \cdot (\mathcal{B}^{Com}_{\mathcal{H}} + \mathcal{B}^{Com}_{\mathcal{S}}).$$

Thus, we have proved Theorem 3.

7 Implementations

We implement the main contribution of Solution 2 - client-side hash renewal under the existing BTS services "OriginStamp" and "Opentimestamps" (The server-side algorithm renewal have been implemented in [11]). In a nut shell, we chose a mp3 file as the data item to be time-stamped, and uploaded the file to the services for three times to simulate the long-term time-stamping process. In each time, the web server calculated the Merkle tree root value and inserted it to a Bitcoin transaction. After the transaction is committed, the web server returned us a time-stamp proof for future verification. The hash function used are all SHA-256 since currently it is secure and applied in the services, but this can be replaced by stronger hash function when SHA-256 is proved weak. The evaluations in the end showing our scheme can be deployed in the existing BTS services easily and efficiently. The details are provided in Appendix A due to the page limit.

8 Conclusions

In this paper, we define the first formal definition and security model for a BLTTS scheme, and analyze that the existing BTS services simply combining with the existing LTB scheme could only prove the existence of data in short-term periods. We observe that for a BLTTS scheme, the security is comprised of two folds: the client-side hash functions and server-side algorithms. A BLTTS scheme must support the cryptographic renewal for both of these algorithms. Then we propose the first BLTTS scheme with three solutions based on different client-side data formats. We analyze that our scheme satisfies the long-term integrity property, and finally we implement our scheme under existing BTS services and found that it is very efficient and easy to deploy it in real applications.

References

1. ISO/IEC 18014-1:2008. Information technology – Security techniques – Time-stamping services – part 1: Framework. Standard, 2008.
2. American National Standard Institute (ANSI). ANSI X9.95-2016 – Trusted Timestamp Management and Security, 2016.
3. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008.
4. Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.
5. Bela Gipp, Norman Meuschke, and André Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. *arXiv preprint arXiv:1502.04015*, 2015.
6. P Todd. Opentimestamps: Scalable, trust-minimized, distributed timestamping with bitcoin. *Peter Todd [Internet]*, 15, 2016.
7. Proof of existence. https://www.proofofexistence.com/.
8. Thomas Hepp, Alexander Schoenhals, Christopher Gondek, and Bela Gipp. Originstamp: A blockchain-backed system for decentralized trusted timestamping. *IT-Information Technology*, 60(5-6):273–281, 2018.
9. Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.
10. Arjen K Lenstra. Key length. Contribution to the handbook of information security. 2004.
11. Long Meng and Liqun Chen. An enhanced long-term blockchain scheme against compromise of cryptography. Cryptology ePrint Archive, Report 2021/1606, 2021. `https://ia.cr/2021/1606`.
12. Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. In *Conference on the Theory and Application of Cryptography*, pages 437–455. Springer, 1990.
13. Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II*, pages 329–334. Springer, 1993.
14. ISO/IEC 18014-2:2009. Information technology – Security techniques – Time-stamping services – part 2: Mechanisms producing independent tokens. Standard, 2009.
15. ISO/IEC 18014-3:2009. Information technology – Security techniques – Time-stamping services – part 3: Mechanisms producing linked tokens. Standard, 2009.
16. Stuart Haber and Pandurang Kamat. A content integrity service for long-term digital archives. In *Archiving Conference*, volume 2006, pages 159–164. Society for Imaging Science and Technology, 2006.
17. T Gondrom, R Brandner, and U Pordesch. Evidence Record Syntax (ERS). *Request For Comments–RFC*, 4998, 2007.
18. Dimitris Lekkas and Dimitris Gritzalis. Cumulative notarization for long-term preservation of digital signatures. *Computers & Security*, 23(5):413–424, 2004.
19. Matthias Geihs, Denise Demirel, and Johannes Buchmann. A security analysis of techniques for long-term integrity protection. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 449–456. IEEE, 2016.
20. Ahto Buldas, Matthias Geihs, and Johannes Buchmann. Long-term secure time-stamping using preimage-aware hash functions. In *International Conference on Provable Security*, pages 251–260. Springer, 2017.
21. Long Meng and Liqun Chen. Analysis of client-side security for long-term time-stamping services. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 28–49. Springer, 2021.
22. Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *White Paper*, 3(37), 2014.

23. Bruno Tavares, Filipe Figueiredo Correia, and André Restivo. A survey on blockchain technologies and research. *Journal of Information Assurance and Security*, 14:118–128, 2019.
24. Hussein Hellani, Abed Ellatif Samhat, Maroun Chamoun, Hussein El Ghor, and Ahmed Serhrouchni. On blockchain technology: Overview of bitcoin and future insights. In *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, pages 1–8. IEEE, 2018.
25. Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. *arXiv preprint arXiv:1906.11078*, 2019.
26. Bela Gipp, Corinna Breitinger, Norman Meuschke, and Joeran Beel. Cryptsubmit: introducing securely timestamped manuscript submission and peer review feedback using the blockchain. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–4. IEEE, 2017.
27. Corinna Breitinger and Bela Gipp. Virtual patent-enabling the traceability of ideas shared online using decentralized trusted timestamping. In *Proceedings of the 15th International Symposium of Information Science*, pages 89–95, 2017.
28. Bela Gipp, Jagrut Kosti, and Corinna Breitinger. Securing video integrity using decentralized trusted timestamping on the bitcoin blockchain. In *Mediterranean Conference on Information Systems (MCIS)*. Association For Information Systems, 2016.
29. Angelos Stavrou and Jeffrey Voas. Verified time. *Computer*, 50(3):78–82, 2017.
30. Guangkai Ma, Chunpeng Ge, and Lu Zhou. Achieving reliable timestamp in the bitcoin platform. *Peer-to-Peer Networking and Applications*, 13:2251–2259, 2020.
31. Gabriel Estevam, Lucas M Palma, Luan R Silva, Jean E Martina, and Martín Vigil. Accurate and decentralized timestamping using smart contracts on the ethereum blockchain. *Information Processing & Management*, 58(3):102471, 2021.
32. Pawel Szalachowski. (short paper) towards more reliable bitcoin timestamps. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 101–104. IEEE, 2018.
33. Yuan Zhang, Chunxiang Xu, Nan Cheng, Hongwei Li, Haomiao Yang, and Xuemin Shen. Chronos+: An accurate blockchain-based time-stamping scheme for cloud storage. *IEEE Transactions on Services Computing*, 13(2):216–229, 2019.
34. Andrew Sward, Ivy Vecna, and Forrest Stonedahl. Data insertion in bitcoin's blockchain. *Ledger*, 3, 2018.
35. Yuefei Gao and Hajime Nobuhara. A decentralized trusted timestamping based on blockchains. *IEEJ Journal of Industry Applications*, 6(4):252–257, 2017.
36. Ilias Giechaskiel, Cas Cremers, and Kasper Bonne Rasmussen. On bitcoin security in the presence of broken crypto primitives. *IACR Cryptol. ePrint Arch.*, 2016:167, 2016.
37. Masashi Sato and Shin'ichiro Matsuo. Long-term public blockchain: Resilience against compromise of underlying cryptography. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8. IEEE, 2017.
38. Fengjun Chen, Zhiqiang Liu, Yu Long, Zhen Liu, and Ning Ding. Secure scheme against compromised hash in proof-of-work blockchain. In *International Conference on Network and System Security*, pages 1–15. Springer, 2018.
39. National Institute of Standards and Technology (NIST). Nist policy on hash functions. Standard, 2017. Online available: https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions.
40. National Institute of Standards and Technology (NIST). Digital signature standard (dss). Standard, 2013.
41. Yuan Zhang, Chunxiang Xu, Hongwei Li, Haomiao Yang, and Xuemin Shen. Chronos: Secure and accurate time-stamping scheme for digital files via blockchain. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

A Implementations

In order to simulate our BLTTS scheme in a real block system, we test the client-side hash renewal in the Solution 2 of our proposed scheme by leveraging the existing BTS services "OriginStamp" and "Opentimestamps". Specifically, we upload a mp3 file called "Blue Moon" to the services at three distinct dates, the services compute the hash value of our file and the Merkle tree root hash value for our file and others, insert the results respectively into three blocks on Bitcoin blockchain, and returns us time-stamp proofs after the blocks are confirmed. Although all of the hash values are determined by the same hash function SHA-256 since now it is secure, we can operate in the same way when SHA-256 becomes weak and a new option is available.

A.1 Time-stamping process

As an example, we illustrate the first time-stamping process implemented on 5^{th} April, 2021 in detail as the following:

After we submit the mp3 file to the OriginStamp service, the returning time-stamp proof is shown as Fig. 6. The title records the submission time of the mp3 file, which is 00:01:32, 5^{th} April, 2021; the string after "Hash" is the hash value of our file computed by SHA-256; the string after "Root Hash" is the Merkle tree root hash value of our file and other files; and the string after "Transaction" is the transaction ID (hash value of the transaction) that indicates the particular transaction containing the "Root Hash". The time-stamp proof is publicly accessible at the website https://www.blockchain.com/explorer by searching block number 677785, or the transaction ID shown on Fig. 6.

Fig. 6: First part of the time-stamp proof

Thereafter, we submit the same file to the Opentimestamp service twice separately at 11:32:06, 9^{th} August, 2021, and 17:02:21, 12^{th} August,2021 to simulate the BTSRen algorithm. The time-stamp proof can be found on block 694946, with transaction

ID "2d3279e70ff2f2b14efce10c95c3b3e72f6c41ad95c538ef91018f70feea446d" and on block 695443, with transaction ID "418e002df581bcb670ebfc2f24f5db7b8d777b9114 f53f32279c979229373d36" respectively.

A.2 Verification

In terms of the verification procedures specified in 5.3, it is straightforward to verify that the hash representation ("Root Hash") of our file is stored with the Bitcoin transaction, the transaction is confirmed in Bitcoin blockchain, and the server-side algorithms under Bitcoin blockchain are currently secure. Then we can also verify that the hash value of the mp3 file and the "Root Hash" value are correctly calculated by using the SHA-256 hash function. At last, every time-stamp proof is generated when the client-side hash function is secure, the file is proved existed at the time displayed on the earliest time-stamp proof. In our experiments, we can prove that the mp3 file "Blue Moon" existed at 01:00, 5th April, 2021 even the SHA-256 hash function is later compromised.

A.3 Evaluation

We evaluate our scheme from the following four aspects: network delay, storage overhead, service fee, and operability.

Network delay. There is a delay between the submission time of the file and the confirmation time of the transaction. In terms our experiments, this delay is around 60 minutes for Bitcoin. For our scheme, if only the user submits the file when the client-side hash function is not actually compromised, the proof of existence of the file is succeed. Based on the SHA-1 breakage example, it will take years from theoretical attacks to a practical attack, thus this delay is acceptable.

Storage overhead. Our implementation of Solution 2 only adds a hash value into the blockchain at once, the overhead depends on the output size of the used hash function. For SHA-256 function, the output size is 256 bits. If the output size of new hash functions increases in the future, such as 512 bits, 1024 bits etc, it will bring bigger overhead at that time. But there are different data insertion methods, some of them allows bigger data size to be submitted. The overhead is manageable if only the output size of the new hash function does not increase to a considerable level.

Service fee. The consumed costs of our scheme depends on which BTS service is used. For Proof of Existence service, the submission of every single file costs 0.25 mBTC $\approx$ 8.3 GBP; the Opentimestamp service is free of charge; and the Originstamp service provides both free service and subscription plans for different levels of service. To be cost-effective, the Opentimestamp and Originstamp services are optimal choices.

Operability. Our scheme only requires users to submit their files to any of the BTS service or with their Bitcoin transactions after they know the hash function is needed to be updated. As the above discussed, it is not required for a very accurate date or time. A user can take only several seconds to submit the file, wait 1 hour to get the time-stamp proof during several years. The operations are simple and efficient.