A Blockchain-Based Long-Term Time-Stamping Scheme

Long Meng(✉) and Liqun Chen

University of Surrey, Guildford, UK
long.meng@surrey.ac.uk

Abstract. Traditional time-stamping services confirm the existence time of data items by using a time-stamping authority. 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 service depends on both the security of the underlying cryptographic algorithms and the reliability and trustworthiness of TSAs.

In reality, TSAs may not always be reliable or trustworthy. If a TSA is compromised, the validity of the time-stamp tokens from this TSA could be

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13554, pp. 3–24, 2022.
https://doi.org/10.1007/978-3-031-17140-6_1

threatened no matter whether the underlying cryptographic algorithms are still secure or not. Therefore, the requirement for 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) This is a decentralized system, so it eliminates the trust requirement on central authorities. (2) A blockchain is tamper-resistant, as transactions are validated by multiple nodes before being stored in a block. Once a block is 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, and inserts the tree root value into a blockchain transaction. The transaction record and the time-stamp in the block become the existence proof of data items. Compared to traditional time-stamping services, BTS services get rid of potential attacks from malicious manipulation or collusion of TSAs. In the popular trends of decentralized applications, BTS services are much better choices than traditional time-stamping services.

A BTS service makes use of hash functions and digital signature schemes to build a blockchain (we collectively call them server-side algorithms), and also uses hash functions to hash users' data (we call them client-side hash functions). 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 period due to the operational life cycle or increasing computational powers of attackers. Particularly, the upcoming quantum computers are considered to break most of the broadly-used signature algorithms and increase the speed of attacking hash functions [9]. However, for many types of digital data, such as identity information, health records, history archives, etc., the existence proof of data needs to be maintained for decades or even permanently, which is much longer than the lifetime of a single cryptographic algorithm.

In this work, if a scheme is secure for a long period that is not bounded by 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 the 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 [10], in which the server-side algorithms could be securely transferred to stronger ones. But our proof shows that the solution is vulnerable to attacks after the client-side hash function 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 the 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 connected in time sequence 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.

2 Related Works

Traditional Time-Stamping. In 1990, Haber and Stornetta proposed the prototype of digital time-stamping with two techniques: linear linking and random witness [11]. In 1993, Bayer et al. proposed a solution for time-stamp renewal [12]: 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. The ideas of [11,12] were designed into a time-stamping system for the Belgian project TIMESEC [13].

In further years, the ideas of [11,12] 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 [12] have been extended into several long-term integrity schemes [16,17], but the security analysis of such schemes was not given, until Geihs et al. formalized this idea separately into a signature-based long-term integrity scheme [18], and a hash-based long-term time-stamping scheme [19]. These two schemes provide substantial frameworks for analyzing the security of long-term time-stamping schemes. However, the works of [18,19] only address the renewal of server-side algorithms, the renewal of client-side hash functions is not addressed.

Besides, Meng et al. found that the ISO/IEC standard [1,14,15] did not specify the renewal of client-side hash functions for traditional time-stamping schemes [20], which causes the schemes could only achieve short-term integrity. Then they proposed and analyzed the first comprehensive long-term time-stamping scheme that allows the renewal of both client-side hash functions and

server-side algorithms [21]. We are inspired by the ideas in [18,19], 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] that leverages the idea of time-stamping [11–13]. 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 [4,23,24]. In our paper, the structure of blockchain shown 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, many applications were built on top of the "OriginStamp" service, e.g., manuscript submission [25], virtual patents [26], secure videos [27]. 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 Sect. 5.

Apart from the design of BTS services, some researchers explored the reliability of the time-stamps included in the blockchain [28–31]. Their research shows that the time-stamps in blockchains are not accurate and could be manipulated for attacks. They proposed distinct solutions to this issue: [30] and [28] had slightly different ideas about leveraging an external TSA since it can provide accurate time records; [31] claimed to integrate the hash value of a user's document with a constant number of latest confirmed blocks on the Ethereum blockchain; [29] proposed to use a smart contract that intermediates between a user and some time-stamp providers according to some selection strategy on the Ethereum blockchain. These ideas can be adopted for reliable and accurate blockchain time-stamps in our proposed scheme.

For the topic of how to insert data into a blockchain, Sward et al. provided a comprehensive survey for inserting arbitrary data into the Bitcoin blockchain [32]. 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 [33], 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. Giechaskiel et al. analyzed the impacts of broken cryptographic primitives on Bitcoin [34]. This work shows that the compromise of SHA-256, RIPEMD160 and ECDSA algorithms in the Bitcoin blockchain may cause the stealing of coins, double spending, repudiated payments, etc. 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 [35], and Chen et al. proposed an improved LTB scheme [36] to avoid the hard fork caused by the hash function renewal in [35] when using a proof-of-work blockchain. Recently, Meng et al. observed that [35,36] only defined the transition from the first algorithm to the second one, and the security of those schemes is not analyzed. Then they proposed an enhanced LTB scheme [10] that enables algorithm renewal in long-term periods, which has been proved secure under their proposed security model. In this work, we borrow the ideas of [10] for achieving server-side algorithm renewal as reviewed in Sect. 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 [24]. 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 value of all block data. The block data contains a list of transactions along with their corresponding digital signatures.

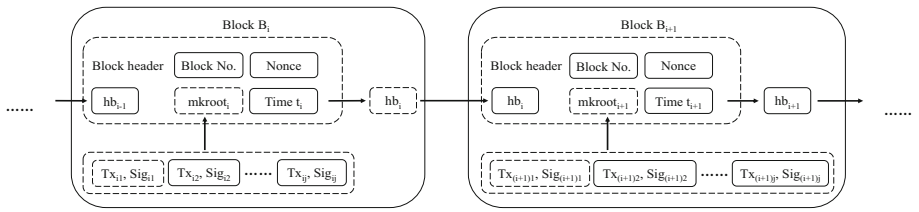

Fig. 1. The general structure of a blockchain

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 $(\mathrm{Tx}_{i1},\ \mathrm{Sig}_{i1}), ..., (\mathrm{Tx}_{ij},\ \mathrm{Sig}_{ij})$ in the block are aggregated together by using a Merkle tree. The resulting root 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, and the chaining of hash values between blocks protects the integrity of block data.

Long-term Blockchain Scheme. For a long-term blockchain (LTB), we review the ideas of the secure LTB scheme proposed by Meng et al. [10], which could be divided into a hash transition procedure and a signature transition procedure.

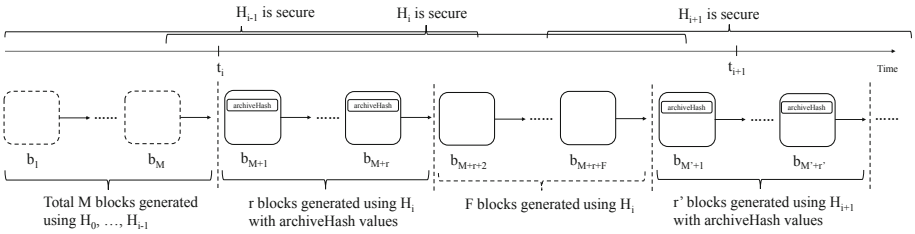

Fig. 2. The hash transition procedure of the LTB scheme proposed by Meng et al.

The hash transition procedure (as shown in Fig. 2) is performed by the blockchain system. Assume at time $t_i (i \geq 1)$ when 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 includes 3 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, i.e., $archiveHash_{i1} = H_i(b_1, ..., b_s)$, ..., $archiveHash_{ir} = H_i(b_{(r-1)s+1}, ..., b_M)$, and stores these $archiveHash$ values separately in the block header of b_{M+1}, ..., b_{M+r}. b_{M+1}, ..., b_{M+r} uses 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 repeat 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 verification procedures of hash transitions check: 1) the correctness of every block (include the merkle tree root value, block hash value, signatures, and $archiveHash$ field etc.), 2) the i-th hash transition happens within the time period that at least hash functions H_{i-1}, H_i are secure, and 3) the latest hash function used in the blockchain is still secure at the verification time.

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. 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. The verification procedures of signature transitions check: 1) the correctness of every block, 2) the i-th signature transition happens within the period that at

least signature schemes S_{i-1}, S_i are secure, and 3) the latest signature scheme used in the blockchain is still secure at the verification time.

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 the 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 stores the data in a block, which provides existence proofs of the data. The verifier checks the validity of the 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 a data item D and a blockchain *blc* as input and 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 some cryptographic algorithms in C_{i-1} is threatened but still secure, the time-stamp renewal algorithm BTSRen takes a data item D and the blockchain *blc* as input and 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 as 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 $b = 0$.

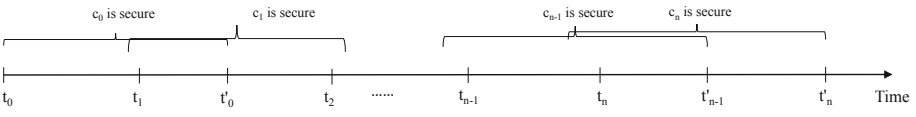

Fig. 3. Timeline of cryptographic algorithm lifetime and renewal

Timeline. Figure 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, both c_{i-1} and c_i are secure. We argue that this renewal time window is reasonable and practical. For example, the SHA-1 algorithm was theoretically broken in 2005 [37], but the first real collision pair of SHA-1 was found in 2017 [38]. The middle 12 years are the renewal window from SHA-1 to SHA-2. 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). 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 the same criteria, e.g., the global time.

4.2 Security Model

In a BLTTS scheme, we make the 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 could prove the existence of data items in long-term periods that are not bounded by 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* $\mathrm{TS}_0,\ ...,\ \mathrm{TS}_n$ *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 can compromise a BLTTS scheme if it could claim that a data item exists at a point in time but it does not exist, or 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 by the lifetimes of underlying cryptographic algorithms.

To formalize this, the long-term integrity model is defined as an experiment, which is displayed as Algorithm 1, running between a long-lived adversary $\mathcal{A}$ and a simulator $\mathcal{B}$. $\mathcal{B}$ has computational resources comparable to $\mathcal{A}$. $\mathcal{A}$ could 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, who returns the corresponding computational power P according to the timeline introduced in Sect. 4.1. That means, P develops with the increase of t but is restricted within each period. The ability that $\mathcal{A}$ can break or cannot break any algorithm depends on P.
2. $Blc(\cdot)$: TS $\leftarrow Blc(x)$, $R \leftarrow R \parallel (x,\ \text{TS})$. $\mathcal{A}$ 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, and meanwhile records x along with TS in a list R.

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

```
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 $\mathbf{Pr}[\mathbf{Exp}^{\text{LTI}}_{\text{BLTTS}}(\mathcal{A}) = 1]$ 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 $\mathcal{B}^{Com}_{\mathcal{H}}$, and the probability that $\mathcal{B}$ breaks at least one signature scheme within its validity period as $\mathcal{B}^{Com}_{\mathcal{S}}$.

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

5 The Proposed BLTTS Scheme

In this section, we first briefly show 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 Sect. 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 depending 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.

Table 1. Notation

$n \in \mathcal{N}$	Number of cryptographic algorithm	D	Data item to be time-stamped
$i \in \{0,\ n\}$	Index of cryptographic algorithm	C_i	i-th cryptographic algorithm tuple
c_i	A particular type of algorithm in C_i	$c.t_i,\ c.t_i'$	Starting and breakage time of c_i
cH_i	i-th client-side hash function	$sH_i,\ S_i$	i-th server-side hash/signature scheme
TS_i	Time-stamp proof using C_i	blc	The blockchain used for time-stamping
t_v	The verification time	h_i	Hash value computed through cH_i
b_i	The block provides TS_i	tx_i	Transaction data
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 in b_i	ts_i	Time-stamp included in block b_i
VD	Verification data used in BTSVer	$pc,\ ps$	Client and server-side hash path
$a \Leftarrow b$	Store parameter b into a	$a \subseteq b$	a is included in b
MT(H; D, p)	MT: Merkle tree aggregation algorithm, H: hash function, D: data, p: hash path		

Existing BTS Schemes. The existing Blockchain-based Time-Stamping (BTS) schemes, e.g., "Proof of existence" [7], "OpenTimestamps" [6] and "OriginStamp" [5], can be summarized as the black fonts in Fig. 4. Since these schemes do not specify the BTSRen algorithm, they do not comply with our BLTTS definition in Sect. 4.1. It is trivial to prove that they are vulnerable to attacks after any of cH_0, sH_0, or S_0 is compromised.

Intuitive BLTTS Solution. As reviewed in Sect. 3, the existing LTB scheme [10] 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 the BTSVer algorithm in the existing BTS schemes by leveraging the LTB scheme (as the red fonts in Fig. 4). Now we analyze the long-term security of the intuitive solution based on our security model proposed in Sect. 4.2.

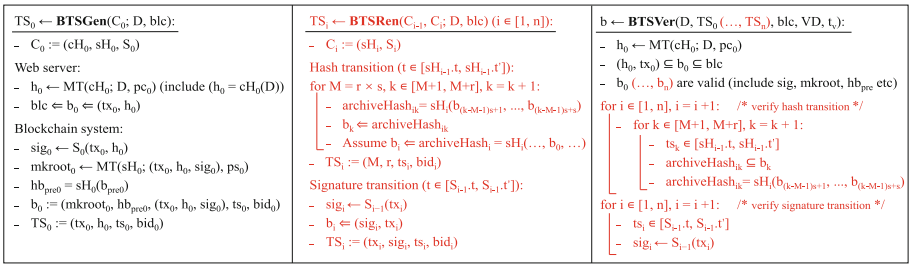

Fig. 4. An Intuitive BLTTS solution that directly combines the existing BTS schemes (black fonts) and an LTB scheme (red fonts) (Color figure online)

Theorem 1. *The intuitive BLTTS solution specified in Fig. 4 does not hold the 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 on 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 that the data item is not exposed to the blockchain after it is hashed. The long-term security on the server side cannot guarantee the long-term security on the client side. So far, a BLTTS scheme does not exist. This motivates us to propose a BLTTS scheme (in Sect. 5.1) that satisfies long-term integrity.

5.1 Proposed BLTTS Scheme with Three Solutions

Roadmap. As discussed before, the LTB scheme [10] only guarantees the long-term security of the server side. The obstacle is the involvement of client-side hash functions. As reviewed in Sect. 2, the ISO/IEC standard has missed the renewal mechanism for client-side hash functions [20]. In [21], this issue has been analyzed and a comprehensive scheme that supports both client-side and

server-side renewal has been proposed for traditional time-stamping. This gives us the following inspirations: 1) the client-side security is easy to be overlooked even by the ISO/IEC standard, 2) client-side and server-side security have the same level of importance and the failure of either side is a bottleneck for long-term time-stamping, and 3) the technique for client-side renewal proposed in [21] could be studied for a BLTTS scheme.

In general, our proposed scheme is composed of two folds. For server-side long-term security, we borrow the LTB scheme from [10]. Then we propose three solutions for achieving client-side long-term security: 1) remove the client-side hash functions, 2) renew client-side hash functions with independent time-stamp proofs, and 3) renew client-side hash functions with connected time-stamp proofs. These solutions are corresponding to the Solution 1, 2, and 3 as presented in Fig. 5. Some parts of the algorithms are referred to Fig. 4.

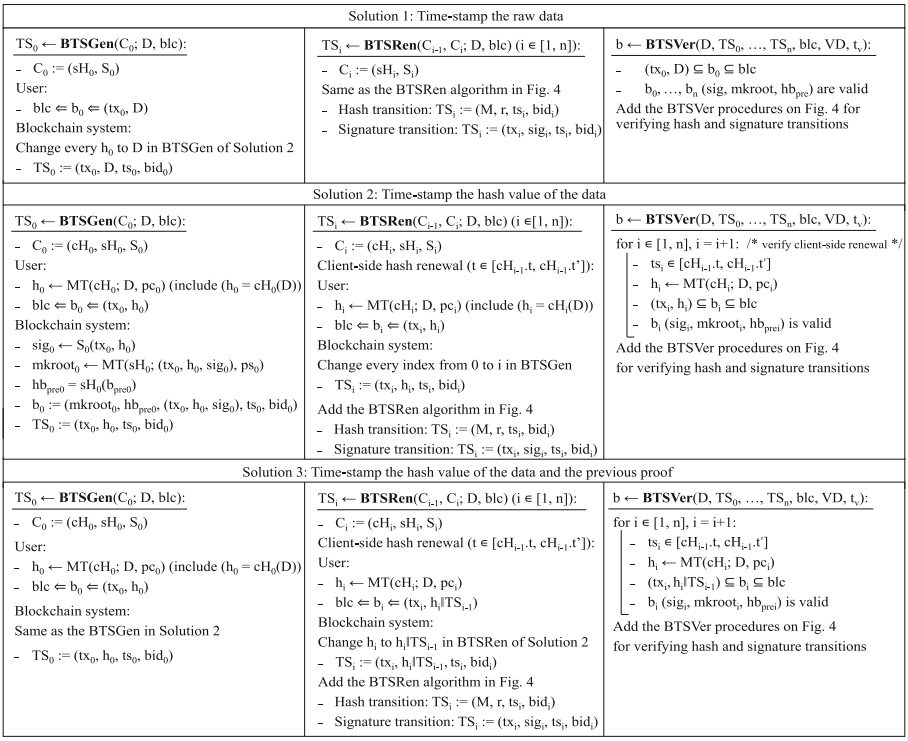

Fig. 5. Proposed BLTTS scheme with three solutions

Remarks

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 that 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 evidence 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 [28–31] 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 [32]. The user should select a data insertion method that has enough capacity for the data item and while it is cost-effective.

5.2 Solutions Comparison

As Table 2 shows, we provide a comparison between Solutions 1, 2, and 3 in the following 6 factors: 1) the renewal type that the user needs to perform, 2) whether the time-stamped data is exposed to the 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.

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

Sols	Cryptographic renewal performed by users	Data exposure	Data size limit per transaction	Costs	Time-stamp connection	Compatible with BTS
1	Server-side signature scheme	Exposed	Limited	Not free	Both sides connected	No
2	Client-side hash function Server-side signature	No	Unlimited	Can be free	Server-side connected	Yes
3	Client-side hash function Server-side signature	No	Unlimited	Not free	Both sides connected	No

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 proofs generated from the 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 a hash value of the data item into a blockchain transaction, thus this solution is not compatible with the services. A user needs to 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 the 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 the 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 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 holds each security property in terms of the security models and definitions in Sect. 4.2.

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 are 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 $h_i \parallel \mathrm{TS_{i-1}}$) 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 $h_i \parallel \mathrm{TS_{i-1}}$), 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. □

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 Sect. 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 the 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 $\mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI,\ C1}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1]$ and $\mathbf{Pr}[\mathbf{Exp}^{\mathrm{LTI,\ C2}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1]$ 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 $\mathcal{B}^{Com}_{\mathcal{S}}$ 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 $\boldsymbol{Pr}[\boldsymbol{Exp}^{\mathrm{LTI,\ C1}}_{\mathrm{BLTTS}}(\mathcal{A}) = 1] \leq c \cdot (\mathcal{B}^{Com}_{sH} + \mathcal{B}^{Com}_{S})$.*

Proof. Since we adopt the existing LTB scheme [10] for server-side algorithm renewal, their proofs show that the LTB scheme satisfies the following two properties:

- Long-term integrity: there is a negligible probability that $\mathcal{A}$ can claim a non-existed data item or tamper with data in any existing blocks on the blockchain without being detected in long-term periods.
- Long-term unforgeability: there is a negligible probability that $\mathcal{A}$ can output 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 [10] 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 $\text{MT}(cH_1; x, pc_1)$ is a part of TS_1. Now repeating the previous situation, $\mathcal{A}$ can compute either $\text{MT}(cH_2; x, pc_2)$ and obtains TS_2 (Case a), or determine $\text{MT}(cH_2; x', pc'_2)$ 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 $\text{MT}(cH_1; x, pc_1) = \text{MT}(cH_1; x', pc'_1)$. 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 has been implemented in [10]). The Opentimestamps deploys the service on Bitcoin, and the OriginStamp implements the service on Bitcoin, Ethereum, and Ayon blockchain for multiple proofs. The results show that our scheme is very practical and efficient to be deployed into a real blockchain. The details are presented in Appendix A.

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 combined 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 be deployed in real applications.

Acknowledgements. This work is supported by the European Union's Horizon 2020 research and innovation program under grant agreement No.779391 (FutureTPM), grant agreement No. 952697 (ASSURED), and grant agreement No. 101019645 (SECANT).

A Implementations

In a nutshell, we chose an mp3 file as the data item to be time-stamped and uploaded the file to the services 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 into a Bitcoin transaction. After the transaction is committed, the web server returned us a time-stamp proof for future verification. The hash functions used are all SHA-256 since currently it is secure and applied in the services, but this can be replaced by stronger hash functions when SHA-256 is proved weak.

Time-Stamping Process. As an example, our first time-stamping process was implemented on 5^{th} April 2021. After we submited the mp3 file to the OriginStamp service, the returning time-stamp proof is shown as the upper part of 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; 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. As shown in the down part of Fig. 6, the "Root Hash" of our file is stored in the OP_RETURN script of the Bitcoin transaction.

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 and block 695443 respectively.

Verification. In terms of the verification procedures specified in Sect. 5.1, it is straightforward to verify that the hash representation ("Root Hash") of our

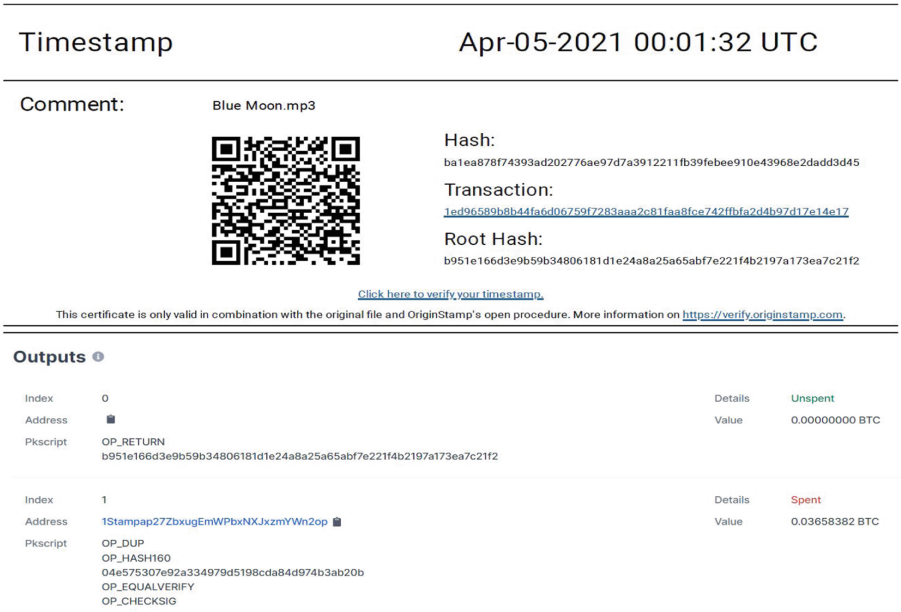

Fig. 6. The returned time-stamp proof from the OriginStamp service (upper) and the OP_RETURN script on the Bitcoin blockchain

file is stored with the Bitcoin transaction, the transaction is confirmed in the 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 is 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 existence of the file is proved 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.

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

Network Delay. In our experiment, the delay between the submission time of the file and the confirmation time of the transaction is around 60 min for Bitcoin. If the user always submits their file for renewal at least 60 min before the current client-side hash function is practically compromised, the long-term existence proof of the file is guaranteed. Considering the breakage of SHA-1 collision-resistance as an example, it took 12 years from the theoretical attack to practical attack, thus this amount of 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 underlying hash function. The output size for SHA-256 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 a bigger overhead. However, there are different data insertion methods, some of which allow bigger data sizes to be submitted. The overhead is manageable as long as the output size of the new hash function does not increase to an unmanageable level.

Service Fee. The consumed costs of our scheme depend on which BTS service is used. For the 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 services 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, and wait 1 h to get the time-stamp proof in several years. The operations are simple and efficient.

References

1. ISO/IEC 18014–1:2008. Information technology - Security techniques - Time-stamping services - Part 1: Framework
2. American National Standard Institute (ANSI). ANSI X9.95-2016 - Trusted TimeStamp Management and Security (2016)
3. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. In: Decentralized Business Review, p. 21260 (2008)
4. Zheng, Z., et al.: Blockchain challenges and opportunities: a survey. Int. J. Web Grid Serv. **14**(4), 352–375 (2018)
5. Gipp, B., Meuschke, N., Gernandt, A.: Decentralized trusted timestamping using the crypto currency bitcoin. arXiv preprint arXiv:1502.04015 (2015)
6. Todd, P.: Opentimestamps: scalable, trust-minimized, distributed timestamping with bitcoin. In: Peter Todd, vol. 15 (2016)
7. Proof of Existence. https://www.proofofexistence.com/
8. Hepp, T., et al.: OriginStamp: a blockchain-backed system for decentralized trusted timestamping. Inf. Technol. **5–6**, 273–281 (2018)
9. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings, ACM Symposium on the Theory of Computing, pp. 212–219 (1996)
10. Meng, L., Chen, L.: An enhanced long-term blockchain scheme against compromise of cryptography. Cryptology ePrint Archive, Report 2021/1606 (2021)
11. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 437–455. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_32

12. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Sequences II, pp. 329–334 (1993)
13. Massias, H., Avila, X.S., Quisquater, J.J.: Design of a secure timestamping service with minimal trust requirement. In: The 20th Symposium on Information Theory in the Benelux (1999)
14. ISO/IEC 18014–2:2009. Information technology - Security techniques - Time-stamping services - Part 2: Mechanisms producing independent tokens
15. ISO/IEC 18014–3:2009. Information technology - Security techniques - Time-stamping services - Part 3: Mechanisms producing linked tokens
16. Haber, S., Kamat, P.: A content integrity service for long-term digital archives. In: Archiving Conference, vol. 2006, no. 1, pp. 159–164. Society for Imaging Science and Technology (2006)
17. Gondrom, T., Brandner, R., Pordesch, U.: Evidence record syntax (ERS). In: Request For Comments-RFC 4998 (2007)
18. Geihs, M., Demirel, D., Buchmann, J.: A security analysis of techniques for long-term integrity protection. In: 2016 14th Annual Conference on Privacy, Security and Trust (PST), pp. 449–456. IEEE (2016)
19. Buldas, A., Geihs, M., Buchmann, J.: Long-term secure time-stamping using preimage-aware hash functions. In: International Conference on Provable Security, pp. 251–260 (2017)
20. Meng, L., Chen, L.: Reviewing the ISO/IEC standard for timestamping services. IEEE Commun. Stand. Maga. **5**, 20–25 (2021)
21. Meng, L., Chen, L.: Analysis of client-side security for long-term time-stamping services. In: Sako, K., Tippenhauer, N.O. (eds.) ACNS 2021. LNCS, vol. 12726, pp. 28–49. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78372-3_2
22. Buterin, V.: A next-generation smart contract and decentralized application platform. In: White Paper, vol. 3, no. 37 (2014)
23. Tavares, B., et al.: A survey on blockchain technologies and research. J. Inf. Assur. Secur. **14**, 118–128 (2019)
24. Yaga, D., et al.: Blockchain technology overview. In: National Institute of Standards and Technology Internal Report (2019)
25. Gipp, B., et al.: CryptSubmit: introducing securely timestamped manuscript submission and peer review feedback using the blockchain. In: 2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL), pp. 1–4. IEEE (2017)
26. Breitinger, C., Gipp, B.: Virtual patent-enabling the traceability of ideas shared online using decentralized trusted timestamping. In: Proceedings of the 15th International Symposium of Information Science, pp. 89–95 (2017)
27. Gipp, B., Kosti, J., Breitinger, C.: Securing video integrity using decentralized trusted timestamping on the bitcoin blockchain. In: Mediterranean Conference on Information Systems (MCIS). Association For Information Systems (2016)
28. Ma, G., Ge, C., Zhou, L.: Achieving reliable timestamp in the bitcoin platform. Peer-to-Peer Netw. Appl. **13**(6), 2251–2259 (2020). https://doi.org/10.1007/s12083-020-00905-6
29. Estevam, G., et al.: Accurate and decentralized timestamping using smart contracts on the Ethereum blockchain. Inf. Process. Manag. **58**(3), 102471 (2021)
30. Szalachowski, P.: (Short Paper) Towards more reliable bitcoin timestamps. In: Crypto Valley Conference on Blockchain Technology, pp. 101–104 (2018)
31. Zhang, Y., et al.: Chronos+: an accurate blockchain-based time-stamping scheme for cloud storage. IEEE Trans. Serv. Comput. **13**(2), 216–229 (2019)
32. Sward, A., et al.: Data insertion in bitcoin's blockchain'. In: Ledger, vol. 3 (2018)

33. Gao, Y., Nobuhara, H.: A decentralized trusted timestamping based on blockchains. IEEJ J. Ind. Appl. **6**, 252–257 (2017)
34. Giechaskiel, I., Cremers, C., Rasmussen, K.: On bitcoin security in the presence of broken crypto primitives. In: IACR Cryptology ePrint Archive (2016)
35. Sato, M., Matsuo, S.: Long-term public blockchain: resilience against compromise of underlying cryptography. In: 26th International Conference on Computer Communication and Networks (ICCCN), pp. 1–8. IEEE (2017)
36. Chen, F., Liu, Z., Long, Yu., Liu, Z., Ding, N.: Secure scheme against compromised hash in proof-of-work blockchain. In: Au, M.H., et al. (eds.) NSS 2018. LNCS, vol. 11058, pp. 1–15. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02744-5_1
37. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005). https://doi.org/10.1007/11535218_2
38. Stevens, M., Bursztein, E., Karpman, P., Albertini, A., Markov, Y.: The first collision for full SHA-1. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 570–596. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_19
39. National Institute of Standards and Technology (NIST). NIST Policy on Hash Functions. Standard (2017)
40. National Institute of Standards and Technology (NIST). Digital Signature Standard DSS. Standard (2013)