This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

Chronos+: An Accurate Blockchain-based Time-stamping Scheme for Cloud Storage

Yuan Zhang, *Student Member, IEEE,* Chunxiang Xu, *Member, IEEE,* Nan Cheng, *Member, IEEE,* Hongwei Li, *Senior Member, IEEE,* Haomiao Yang, *Member, IEEE,* and Xuemin (Sherman) Shen, *Fellow, IEEE*

Abstract—We propose Chronos+, an accurate blockchain-based time-stamping scheme for outsourced data, where both the storage and time-stamping services are provided by cloud service providers. Specifically, Chronos+ integrates a file into a transaction on a blockchain once the file is created, which guarantees the file's latest creation time to be the time when the block containing the transaction is appended to the blockchain. A sufficient number of consecutive blocks that are latest confirmed on the blockchain is embedded into the file at the creation time. These blocks serve as a time-dependent random seed to prove the earliest creation time, due to blockchains' *chain quality* property. Chronos+ makes the file's timestamp corresponding to a time interval formed by the earliest and latest creation times which are derived from the heights of the corresponding blocks. Due to blockchains' *chain growth* property, such a height-derived timestamp can ensure that the time intervals' range is within a few minutes so as to guarantee the accuracy. We also point out potential threats towards outsourced time-sensitive files and present security analyses to prove that Chronos+ is secure against these threats. Comprehensive performance evaluations demonstrate the efficiency and practicality of Chronos+.

Index Terms—Time-stamping, blockchain, cloud storage, digital investigations.

1 Introduction

With the digital data being explosively generated in recent years, people are increasingly outsourcing their data to cloud servers [2], [3], [4]. Cloud Storage services free people from deploying and maintaining local storage devices and enables them to access the outsourced data from different devices and places via the Internet [5], [6], [7]. In general, the cloud service provider needs to maintain the outsourced data for a prolonged period of time for data archiving [8], [9], [10], [11] and keep track of when the file was created for post investigations. For example, in cloud-based intellectual property systems, a file when an inventor first applied for a patent should be maintained during the lifetime of the patent to determine the patent term. More crucially, this file also serves as a key evidence to indicate who is the first inventor of the patent when a dispute arises [12]. However, it is usual that the defendant argues about the timeliness of the digital evidence during the trial. The defendant might dispute the creation time of the file which is admitted by investigators and utilized for the judgment is not the actual one in reality. Since all files are outsourced to cloud servers, an adversary may incentivize cloud service providers to back-date/forward-date the target file to increase his profits in the system.

Cryptographic time-stamping is the most important way to certify when a file was created in digital investigations. Existing schemes [13], [14], [15], [16] utilize a trusted service provider (TSP) to assist users in time-stamping their files, where a file is transmitted to TSP once it is created, and TSP time-stamps it and responses the file owner with the timestamp. Integrating this mechanism into cloud storage services can securely time-stamp outsourced files. However, there are two problems in such a system:

1) Existing schemes [13], [14], [15], [16] bear a strong assumption that TSP is reliable and honest. Once TSP is compromised, the recorded timestamps can be arbitrarily modified, and the security of these schemes is broken. As such, TSP becomes a single point of failure in these systems.
2) With the deployment of TSP, users' interaction pattern in cloud storage systems is changed: since TSP is a trusted entity that is independent of the cloud service provider, the users are required to interact with both TSP and cloud server to secure their files in data outsourcing. Consequently, users have to bear not only an additional communication burden but also extra costs to employ TSP.

On the other hand, a secure time-stamping scheme can be extended from current blockchain-based storage schemes (e.g., Blockstack [17] and Catena [18]) to address the single-point-of-failure problem and to free from heavy costs to employ TSP. Specifically, with the introduction of the blockchain, the cloud server can provide both the storage and secure time-stamping services: Once a file is generated, it is outsourced to the cloud server. Then, the file

- *Y. Zhang, C. Xu, H. Li, and H. Yang are with the Center for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, China (e-mail: ZY_LoYe@126.com; chxxu@uestc.edu.cn; hongweili@uestc.edu.cn; haomyang@uestc.edu.cn).*
- *N. Cheng is with the School of Telecommunication, Xidian University, China (e-mail: dr.nan.cheng@ieee.org).*
- *X. Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Canada (e-mail: sshen@uwaterloo.ca).*

A preliminary version [1] of this paper was presented at the 2019 IEEE International Conference on Communications (ICC' 19).
The corresponding authors are Chunxiang Xu and Yuan Zhang.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

is integrated into a transaction on a public blockchain (i.e., Bitcoin [19]). After the transaction is recorded into a block on the blockchain, the file has been time-stamped. The block's timestamp that indicates the time when the block was added to the blockchain can serve as the file's timestamp to show that the file was generated earlier than this point in time. Since the blockchain is resistant to modification, anyone cannot back-date/forward-date the recorded file. Nevertheless, this scheme cannot be directly adopted in practice, due to the following reasons. First, Catena [18] and Blockstack [17] are constructed on Bitcoin [19], the timestamp of a block on the Bitcoin blockchain may suffer from up to two-hour errors[1]. Second, there is a gap between the file's creation time and the time that the block containing the transaction is appended to the blockchain. Due to the limited handling capacity of the Bitcoin system [20], a transaction cannot be recorded into the Bitcoin blockchain immediately once it is created, in which it takes about 1 hour for a transaction to be accepted into a mined block in extreme cases. Third, the block's timestamp can only indicate the latest creation time of the file, which fails to prove the earliest creation time of the file. Thus, such a scheme cannot accurately indicate when the file was created.

In this paper, we present an accurate blockchain-based time-stamping scheme dubbed Chronos$^+$ for cloud storage systems[2]. In Chronos$^+$, both the time-stamping and data outsourcing services are provided by a Chronos$^+$ log server which is subject to a cloud service provider. When the log server receives a time-stamping request on a file from a user, it conducts a transaction that integrates the file on a public blockchain. To overcome the long delay to conduct a transaction in Bitcoin, we construct Chronos$^+$ on a more expressive public blockchain system, i.e., Ethereum [21], [22]. As such, the physical time when the block is appended to the Ethereum blockchain can be considered as the file's latest creation time. The key idea behind Chronos$^+$ to accurately determine the physical time when the block is appended to the blockchain is to derive the time from the block's height, due to the fact that the height of blockchain would be steadily increased with respect to both short and long terms, which is formalized as the *chain growth* property of blockchains [23]. To indicate the earliest creation time of the file, a sufficient number of consecutive blocks that are latest confirmed on Ethereum is embedded into the file on the user side before outsourcing. Due to the *chain quality* property of blockchains [24], [25], these blocks cannot be fully controlled by an adversary, and thereby can serve as a *time-dependent random seed* to enable users to prove the earliest creation time of the file. As such, the file's timestamp in Chronos$^+$ is a time interval formed by its earliest and latest creation physical times, where Chronos$^+$ enables the range of the time interval to be few minutes. Besides, with a repaid growth in user demands, a foreseeable increase of time-stamping requests from different users might be sent to the log server. As individual time-stamping of these growing requests could be cumbersome, we further extend Chronos$^+$ to support batch time-stamping, where the log server can efficiently handle multiple requests from different users simultaneously.

Specifically, the contributions of this paper are summarized as follows.

- We analyze the characteristics of outsourced files that need to be protected by time-stamping schemes. With the analysis, we point out potential threats towards time-stamping schemes for cloud storage systems. We also introduce a concept of window of time-stamping (WoT) to measure the practicality of time-stamping schemes.
- We present an accurate, secure, and scalable time-stamping scheme called Chronos$^+$ based on the Ethereum blockchain. Chronos$^+$ is suitable for cloud storage systems, since it enables outsourced files to be time-sensitive without changing the user's interaction pattern. Chronos$^+$ proves that a file was created during a time interval formed by its earliest and latest creation times which are derived from the heights of corresponding blocks and serve as the timestamp of the file. We also extend Chronos$^+$ to support batch time-stamping where multiple time-stamping requests from different users can be performed simultaneously in an efficient manner.
- We provide security analysis to prove that Chronos$^+$ is secure against existing attacks. We give a comprehensive performance evaluation, which demonstrates that Chronos$^+$ is efficient in terms of computation, communication, and monetary costs and WoT. We also conduct experiments on mobile devices to show that Chronos$^+$ can be deployed for mobile users.

The remainder of this paper is organized as follows. In Section 2, we review the related work. In Section 3, we present preliminaries. We define the system model, threat model, and design goals in Section 4. In Section 5, we overview Chronos$^+$. In Section 6, we propose Chronos$^+$. We analyze the security of Chronos$^+$ in Section 7 and evaluate its performance in Section 8. Finally, we draw the conclusions and outlook the future work in Section 9.

2 Related Work

The time-stamping technique plays an important role in protecting digital files, especially in digital investigations. In time-stamping schemes, a file is time-stamped once it is created such that it can be time-sensitive. The security of the time-stamping ensures that anyone (including the file owner) cannot back-date and forward-date the file.

The problem of time-stamping digital files is first introduced by Haber et al. [13], where the first time-stamping scheme was proposed. In this scheme, a trusted service provider (TSP) is introduced to provide the time-stamping service for its users. The key observation behind the scheme in [13] is that the sequence of users requesting timestamps and the files they want to time-stamp cannot be known in advance. With this observation, Haber et al. utilized a hashchain to link a sequence of files from different users via a secure hash function. However, this scheme bears a strong assumption that TSP is reliable. Once an adversary colludes

1. https://en.bitcoin.it/wiki/Block_timestamp

2. In the preliminary version [1], the proposed scheme is called Chronos. In this extended version, we enhance Chronos in terms of security and efficiency, which is expressed by the symbol "+".

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

with TSP, a new hashchain can be regenerated such that an existing file can be re-time-stamped. Following the work proposed by Haber et al., some schemes [14], [15] were proposed with enhanced efficiency. However, the fundamental issue of trusting TSP still exists in these schemes.

In the last decade, we have witnessed the explosive generation of digital data, which causes users the data management problem [26], [27], [28], [29]. Currently, users always prefer to utilizing cloud storage servers to manage their data. Data outsourcing makes the guarantee of data timeliness more challenging than ever, and thereby requires a new time-stamping mechanism for outsourced files. Recently, blockchains have been envisioned as a powerful tool to enhance cloud storage services in terms of security and efficiency [30], [31], [32], [33], [34], [35]. Interestingly, the hashchain in [13] serves as a fundamental primitive for Bitcoin [19] which is the most prominent manifestation of blockchains. Some works have investigated how to construct secure time-stamping schemes on public blockchains that frees from the trusting of TSP. Typical schemes include universal hash time [36] and cryptographic time-stamping through sequential work [37]. The key idea behind these schemes is to enable all files from different users to form an authenticated data structure (e.g., hashchain and Merkle hash tree) with the aid of a public blockchain (e.g., Bitcoin). The timeliness of these files in the data structure is reflected in the chronological order. By doing so, one can determine that a file was generated no earlier than the previous one and no later than the subsequent one. However, these schemes have some issues in practice. First, the timestamp of a file depends on other files in the system, which cannot accurately reveal the physical time when the file was created. Second, these schemes are designed for traditional storage systems where users store their files locally and cannot be directly adopted in cloud storage systems.

Another line of work focuses on designing blockchain-based secure storage systems [38], [39], which can be extended to support secure time-stamping. Typical schemes include Blockstack [17] and Catena [18]. The key technique behind these schemes to support time-stamping is to integrate each file into a transaction on a public blockchain. After the transaction is recorded into a block, the block's timestamp can serve as the file's timestamp. However, such a mechanism is also confronted with two problems. First, the block's timestamp on public blockchains is not accurate, which may suffer from an unacceptable time error. For example, in Bitcoin, the time error may be up to two hours. Second, the timestamp can only prove that the file was generated no later than a point in time.

In the previous version of this paper [1], a secure time-stamping scheme for outsourced files via a public blockchain, dubbed Chronos, has been proposed. In Chronos, the accuracy of a file' timestamp is improved by deriving the timestamp from the corresponding block's height on the blockchain. The earliest creation time of a file is proven by utilizing the chain quality of the public blockchain. Compared with Chronos [1], *we have made the following improvements in this paper*.

- We first analyze existing schemes (including the preliminary scheme in [1]) and point out that they are vulnerable to a type of adversaries called malicious competitors. Specifically, in existing schemes, if a malicious user colludes with the time-stamping service provider, he can launch an attack of "stealing the thunder" by changing the ownership of a target file.
- We propose a time-stamping scheme called Chronos$^+$ based on Ethereum to enhance both the security and efficiency. In particular, Chronos$^+$ is secure against malicious competitors and supports batch time-stamping where the service provider can serve multiple users simultaneously in an efficient way.
- We introduce the concept of WoT to measure the practicality of secure time-stamping schemes. We evaluate the performance of Chronos$^+$, which proves that Chronos$^+$ is efficient. We also conduct the experiment on mobile devices to demonstrate that the high efficiency of Chronos$^+$ on the user side.

3 PRELIMINARIES

3.1 Notations and conventions

We use ℓ to denote the security parameter. Given a finite set S, $|S|$ denotes the number of elements in S. Given two bit-strings x and y, we denote by $x||y$ their concatenation.

3.2 Digital signature and bilinear map

Digital signature. A digital signature scheme enables a signer who has established a public key to sign a message using the corresponding private key. Anyone who obtains the public key can check that the message originated from the signer and was not modified. We utilize BLS signature algorithm [40], [41] to construct Chronos$^+$, since it is considerably shorter and is faster to compute compared with other signature algorithms. BLS signature is constructed on the bilinear map described below.

Bilinear maps. We assume that G is an additive group whose order is a prime p and G_T is a multiplicative group with the same order. $e : G \times G \to G_T$ is a bilinear map if it has three properties: (i) Bilinearity: $e(xP, yQ) = e(P, Q)^{xy}$, $\forall\ P, Q \in G$ and $a, b \in Z_p^*$; (ii) Non-degeneracy: $\forall\ P, Q \in G$ and $P \neq Q$, $e(P, Q) \neq 1$; (iii) There is an efficient algorithm to compute e.

3.3 Blockchain

A blockchain consists of multiple data elements that are called blocks. A group of participants [42], [43] maintains the blockchain, where all blocks are linked to form a chain and secured utilizing a secure hash function. The participants aim at achieving a consensus on the new blocks and securely appending them to the blockchain without trusting each other. Blockchains are publicly verifiable and inherently resistant to modification [44], [45]. They can be mainly classified into two types: private blockchains (including consortium blockchains) and public blockchains. In private blockchains, the participants should be authorized by a centralized authority who can be considered as the owner of the blockchain. Here, the authority can consist of multiple parties, in this case, the blockchain is called consortium blockchain. In public blockchains, the participants can freely join and leave the systems without others' permission.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

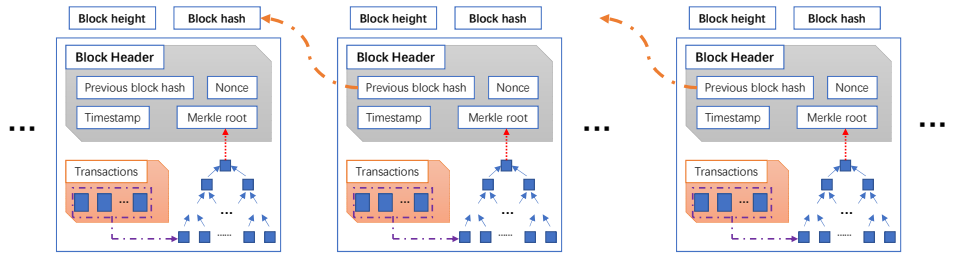


Fig. 1. A simplified Ethereum blockchain

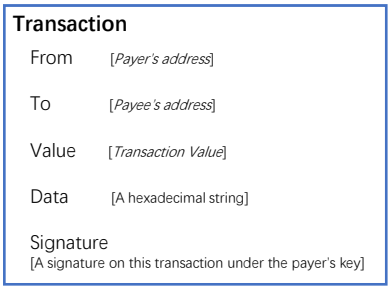


Fig. 2. A graphical transaction in Ethereum

Public blockchains serve as a key component in decentralized cryptocurrencies, e.g., Bitcoin [19] and Ethereum [21]. Essentially, in these cryptocurrencies, the blockchain is used to record a public ledger to keep track of the ownership of each underlying value token. A transaction can be considered a function that changes the ownership of specific tokens and updates the ledger. The participants who maintain the blockchain and add new blocks containing transactions are called *miners*. The security of blockchains ensures that only valid transactions can be recorded. Consensus algorithms play a key role in blockchain systems. Currently, public blockchain systems can be based on multiple consensus algorithms, e.g., proof-of-work (PoW) [19], [21], proof-of-stake (PoS) [46], proof-of-space [47], etc. In this paper, we construct the scheme on a well-established and widely-used (PoW-based) public blockchain, i.e., Ethereum, since it is more expressive than other public blockchains.

A simplified Ethereum blockchain is illustrated in Fig. 1. A block consists of two parts of data. The first one is called the block header which is used to compute the hash value of the block. It includes the following data fields.

- Hash value of the last block. It serves as a pointer to point to the previous block, such that all blocks form a chain.
- Nonce. It is a solution of the PoW puzzles. The miner, who is the first one that finds a valid nonce, can determine and publish this block.
- Timestamp. It is a physical time that indicates when the corresponding block was created. In reality, this timestamp is not accurate, since others would not verify its accuracy. In Ethereum, the timestamp would suffer from up to 900-seconds errors.
- Merkle root. It is the root value of a Merkle hash tree computed from all transactions in the current block.

The second one is called the transaction data, which includes all transactions in the current block. A graphical transaction in Ethereum is shown in Fig. 2. In Ethereum, the ledger can be thought of as a state transition system, where there is a "state" consisting of the ownership status of all existing Ethers (the value token of Ethereum) and a function that takes a state and a transaction as input, and outputs a new state which is the result. The state is composed of accounts which are addresses in the network. Generally, Ethereum supports two types of accounts: externally owned accounts and contract accounts. Contract accounts are controlled by the corresponding contract code. One can invoke the contract by transferring Ethers to the corresponding contract account. Externally owned accounts are controlled by the corresponding private keys. Anyone who has a private key can transfer Ethers from the corresponding externally owned account to other accounts (including the externally owned ones and contract ones). In the transaction, there is a data field. Ethereum enables the user who creates the transaction to set an arbitrary binary string on the data field. More technique details can be found in [19], [21].

From the perspective of cryptography, the blockchain can be considered as a cryptographic protocol that is executed among multiple participants. Such a protocol has the following three fundamental properties [23], [24], [25], [48]:

- φ-chain consistency. At any point, the blockchains in two honest participants can differ only in the last φ blocks with overwhelming probability in φ.
- (ι, φ)-chain quality. In an honest party's blockchain, the fraction of blocks mined by honest parties in any sequence of φ-successive blocks is at least ι.
- Chain growth. The blockchain height would be increased steadily with respect to both short term and long term.

4 Problem Statement

4.1 System model

As shown in Fig. 3, there are three entities in Chronos$^+$: User, Chronos$^+$ log server, and authenticated auditor.

- User. The user is the owner of data files, she/he generates new files and requests time-stamping services from the Chronos$^+$ log server. Chronos$^+$ supports batch time-stamping such that multiple files generated by different users at the same time can be time-stamped simultaneously. We stress that different users are independent in the batch time-stamping, and it would not require any additional operations or investments on the users' devices.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

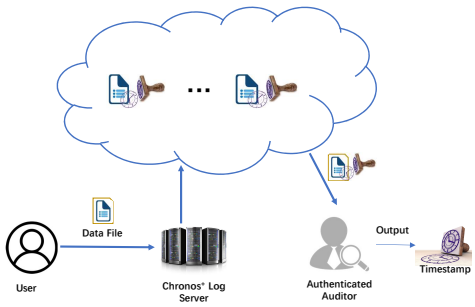


Fig. 3. System model

- Chronos$^+$ log server. The log server provides users both data outsourcing and time-stamping services. In reality, as users prefer to outsource their files to cloud servers, the log server can be subject to cloud service providers. We stress that Chronos$^+$ can also be applied in traditional storage systems where users store their data locally. In this case, the log server is subject to the user herself/himself.
- Authenticated auditor. The authenticated auditor can access the data files stored on the log server and outputs a timestamp for a specific file, which indicates the physical time when the file was generated.

Characteristics of outsourced files protected by time-stamping schemes. Generally, files to be time-stamped are tamper-unpredictable and should be archived timely. It means that when a file is created, the file creator (i.e., file owner) cannot establish the necessary knowledge and motivation to tamper with the timestamp of the file [13]. Meanwhile, the file should be outsourced and time-stamped timely, and the log server needs to well maintain the file and its timestamp to archive data for post investigations. Consequently, the time when the file was accepted by the log server can be considered as the creation time of the file.

Chronos$^+$ impliedly supports the case of multiple log servers, since the auditor can output the physical time when a file was generated. In particular, given two files and the corresponding timestamps, the physical time when these two files were generated can be kept consistent no matter which log server they are store on. For the sake of brevity, we only show the single log server in Chronos$^+$.

4.2 Threat model

As discussed in Section 4.1, the files Chronos$^+$ protects are tamper-unpredictable and should be archived timely [7], [12]. It means that the user cannot establish any motivation to stop or procrastinate on time-stamping the newly generated files. In other words, the user would request the time-stamping service from the log server at the time a file is created. The user would honestly follow this process to guarantee the normal archiving service.

In reality, time-stamping schemes are confronted with threats from two different angles: malicious users and rational log server.

Malicious users. There are two types of malicious users.

- *Malicious file owner*. After the files are outsourced and timestamped, some of them are found to be incriminating and arguable. As such, a file owner may attempt to back-date/forward-date existing files to increase his profits in the system. For example, in a digital investigation, a malicious file creator may attempt to tamper with existing timestamps of files to cover up his wrongdoing. We stress that the file owner would not be misbehavior when she/he requests the timestamp of a file from the log server, since the file owner cannot establish the necessary knowledge and motivation to tamper with the timestamp of the file at this time and the normal archive server would be impeded by such a misbehavior.
- *Malicious competitor*. A malicious competitor is a valid user in the system. He targets at a specific group of users in the system to "steal the thunder". For instance, in a cloud-based intellectual property system, when a user uploads a patent to the log server, a malicious competitor can intercept and tamper with the patent to change the ownership of the patent. This problem is further exacerbated by the fact that the malicious competitor can incentivize the log server to perform the above attack. We stress that any external adversary (e.g., malicious hackers) can become a valid user in the system to increase the advantage. Therefore, an external adversary can be considered as a malicious competitor.

Rational log server. We consider the log server as a rational entity. Since the log server is subject to a cloud service provider, this assumption inherits the one in existing secure cloud storage systems [12], [49], [50], [51]. Particularly, the log server will only deviate from the scheme if its profits can be increased. It can be incentivized by malicious users (including file owners and competitors) to back-date/forward-date the outsourced files. Different from existing works [13], [14], [15], [16], in Chronos$^+$, we assume that any adversary might collude with the log server to modify the files' timestamps.

In our threat model, we also assume that the budget of adversaries is limited, since an adversary who has an unlimited budget can break the security of any blockchain system. Furthermore, since the auditor is subject to authorities and its inputs are publicly verifiable, the correctness of timestamps generated by the auditor can be easily proven. Therefore, we would not consider malicious auditors.

4.3 Window of time-stamping

With the system model described in Section 4.1, in order to assess the practicality of secure time-stamping schemes, we introduce a concept of window of time-stamping (WoT).

Definition 1. Window of time-stamping (WoT) is a time interval from the time that a request on a file's timestamp is made to the time that the timestamp is securely recorded.

Since the longer WoT, the larger the time errors in the timestamp and the longer the latency period that a user has to bear for time-stamping a file. WoT is one of the most significant factors to affect the practicality of secure time-stamping schemes. Therefore, the shorter WoT is, the more practical a time-stamping scheme is.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

4.4 Challenges and design goals

In this paper, we target at designing a blockchain-based accurate time-stamping scheme, in which there exist three challenges:

1) How to ensure the accuracy of files' timestamps. Since the timestamp of a block in Bitcoin might suffer from up to two-hour errors, secure time-stamping schemes that are derived from Bitcoin-based storage systems [17], [18] cannot guarantee the accuracy of files' timestamps. This problem cannot be well addressed by only substituting Bitcoin by other blockchain systems, because existing public blockchain systems do not provide a mechanism to ensure the accuracy of blocks' timestamps. Therefore, how to ensure the accuracy of files' timestamps is a challenging problem.
2) How to prove the earliest creation time of files. Existing blockchain-based time-stamping schemes, such as universal hash time [36] and the scheme in [37], only can prove that a file was generated no earlier than another file. A straightforward way to prove the earliest creation time of a file is to require the file owner to embed creation time in the file before outsourcing to the log server. However, such the strategy requires synchronization among all users and all log servers. This would cause a heavy burden on users, especially for those who equip low-power devices. Thus, how to prove the earliest creation time of files in an efficient and non-repudiation way is very challenging.
3) How to resist malicious competitor. In the threat model of existing schemes, malicious competitors are not considered. However, a malicious competitor might target specific users to "steal the thunder". Note that encrypting the files to be time-stamped cannot resist malicious competitors, since they can collude with the log server to compromise the target files. Hence, how to thwart malicious competitors without sacrificing performance and functionality is a challenge.

To support secure time-stamping under the aforementioned model, three objects should be achieved.

- Efficiency: Time-stamping files should not introduce heavy computation and communication costs on both the log server and users. The log server should be able to handle multiple tasks from different users simultaneously. WoT should be as short as possible.
- Functionality: The scheme should prove that a file was generated during a time interval. The accuracy of timestamp should be ensured and the range of time interval should be kept as short as possible.
- Security: After a file is time-stamped, neither the file owner nor the log server can modify the recorded timestamp, even if they collude with each other. A malicious competitor cannot break the security.

5 Overview of Chronos+

Before presenting Chronos+, we start with a plain blockchain-based time-stamping scheme which is derived from existing blockchain-based secure storage systems(e.g., Catena [18], Blockstack [17], Tierion[3], and Factom[4]). As shown in Fig. 4, the plain scheme works as follows.

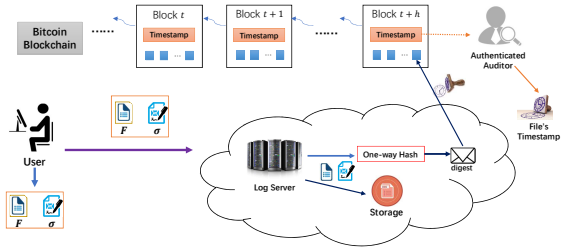


Fig. 4. Plain blockchain-based time-stamping scheme

A user creates a file F and computes the corresponding signature σ. Then she/he sends $\{F, \sigma\}$ to the log server. The log server creates a transaction in Bitcoin and uploads the transaction to the Bitcoin blockchain, where $\{F, \sigma\}$ is recorded into the transaction[5]. After the transaction is confirmed, the log server maintains F, σ, and other auxiliary information (e.g., the information about the block and the transaction containing the data) locally. The timestamp of the block containing $\{F, \sigma\}$ serves as the file's timestamp.

In the plain scheme, the accuracy of the timestamp is arguable, which cannot accurately indicate when the file was created. The inaccuracy is mainly caused by two reasons.

1) The block's timestamp on the Bitcoin blockchain is confronted with up to two-hour errors.
2) A transaction has to wait to be lumped into a block with a considerably long delay (about 1 hour and more in extreme cases).

Improving the accuracy. To guarantee the accuracy of the file's timestamp, an alternative way to obtain the timestamp should be investigated. The key observation behind Chronos+ is that the blockchains' property of *chain growth* can actually be utilized to construct a "clock" indicating the time that each block was chained to the blockchain. Specifically, chain growth formalizes blockchains' property that a blockchain's height will steadily increase in respect of both short and long terms. This enables us to accurately derive the physical time when a block was appended to the blockchain from the block's height on the blockchain. Such the height-derived timestamp overcomes the time errors in the block's timestamp. To avoid the considerably long delay caused by uploading the transaction to the Bitcoin blockchain, we built Chronos+ on Ethereum, since the handling capacity of Ethereum is stronger than those of Bitcoin. This reduces the delay of uploading transactions significantly.

Proving the earliest creation time. The above schemes follow the same paradigm: the file's timestamp *only* corresponds to the time when the block containing the file was appended to the blockchain. However, such a timestamp only can prove that the file was created earlier than the time

3. https://tierion.com/
4. https://www.factom.com/
5. Catena [18] is constructed on the Bitcoin blockchain [19], and utilizes the TOX mechanism to enable multiple data files (even if they are generated by different users) to be chained. Furthermore, Catena employs the "OP-RETURN" outputs to store the data in the Bitcoin blockchain.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

when the block was appended. This would be insufficient for protecting time-sensitive data in reality, especially when malicious competitors exist. A trivial method to prove the earliest creation time of a file is to let the user embed the creation time in the file before sending to the log server. Whereas, this requires a precise time synchronization among all users and all log servers[6], and would introduce heavy communication and computation costs.

We resolve this deadlock by utilizing (ι, φ)-chain quality of blockchains. Instead of requiring the user to embed physical time in the file to prove the earliest creation time, the user only embeds a time-dependent random seed in the file for this purpose. The time-dependent random seed is unpredictable and unforgeable, which proves that the seed was generated no earlier than a point in time. We observe that the hash values of φ-successive blocks on the blockchain can actually serve as such a time-dependent random seed. With the integration of the hash values of φ-successive blocks that are latest confirmed on the Ethereum blockchain, Chronos$^+$ enables users to prove that the file was generated no earlier than the physical time that the last block of φ-successive ones was appended to the blockchain. Again, this physical time is also derived from the block's height to ensure its accuracy. Therefore, the file's timestamp in Chronos$^+$ is a time interval denoted by $[ts_1, ts_2]$, where ts_1 is extracted from the height of the last block in φ-successive ones that are latest confirmed on the Ethereum blockchain when the file was created, and ts_2 is extracted from the height of the block containing the file.

Resistance against malicious competitors. There is still a subtle security problem. Consider a malicious competitor who targets at a specific user for stealing her/his file. To this end, the malicious competitor intercepts the file sent from the user, compromises the user's network, changes the ownership of the file (e.g., changing the author information), and sends the modified file to the log server. By doing so, the malicious competitor is able to steal the user's thunder. Note that such an attack cannot be thwarted by encrypting the file on the user before outsourcing, since the contents of files protected by time-stamping schemes should always be publicly verifiable, and the malicious competitor may incentivize the log server to perform the attack.

We address this issue by an elaborate mechanism of "unlock on delivery". Specifically, the user encrypts the file (using a symmetric encryption algorithm) and sends the ciphertext to the log server. The log server time-stamps the ciphertext as described before. After the timestamp is generated, the user sends the encryption/decryption key to the log server. The log server then decrypts the ciphertext and stores the file as well as the encryption/decryption key and the timestamp locally. The "stealing the thunder" attack no longer works, because the malicious competitor cannot change the ownership of the file without the encryption/decryption key, even if he colludes with the log server.

Security against equivocation attacks. In reality, a malicious user may collude with the log server to perform equivocation attacks to modify an existing timestamp of a specific file that generated by himself. In particular, after a file is outsourced and the corresponding information is recorded into a transaction on the blockchain, the file owner is able to incentivize the log server to "re-time-stamp" the file and substitute the newly generated timestamp for the existing one. As such, the malicious user can equivocate on the timestamp of the target file. It is cumbersome to detect such the attack in practice, since it requires an auditor to maintain the entire blockchain and scan it to ensure the non-equivocation. The plain scheme resists the attack by utilizing a non-membership proof which is constructed on the Bitcoin's UTXO mechanism [18]. However, Chronos$^+$ cannot follow this strategy to resist equivocation attacks, since Ethereum does not use the UTXO mechanism.

To guarantee the non-equivocation, we construct a new non-membership proof on the Ethereum's blockchain. The accounts used to create the transactions in Chronos$^+$ are specially-crafted and dedicated, which enables an authenticated auditor to check whether the number of transactions created by the account matches the number of files that have been time-stamped. This number can be easily extracted from the "nonce" field of the account in Ethereum. This yields our final scheme which provides a secure, accurate, and efficient time-stamping service for users.

6 The Proposed Chronos$^+$

6.1 Construction of Chronos$^+$

A user $\mathcal{U}$, a log server $\mathcal{LS}$, and an authenticated auditor $\mathcal{A}$ are involved in Chronos$^+$. Fig. 5 shows Chronos$^+$. We describe next the four algorithms of Chronos$^+$, **Setup**, **Outsource**, **TimeStamp**, and **CheckStamp**.

Setup. With the security parameter ℓ, the system parameters $\{p, P, G, G_T, e, h, H, A_{\mathcal{LS}}, A_{\mathcal{U}}, \mathsf{E/D}\}$ are determined, where G is an additive group whose generator is P with a prime order p, G_T is a multiplicative group with the same order, $e : G \times G \to G_T$ is a bilinear pairing, $h : \{0,1\}^* \to Z_p$, $H : \{0,1\}^* \to G$, $A_{\mathcal{LS}}$ is the $\mathcal{LS}$'s account and $A_{\mathcal{U}}$ is the $\mathcal{U}$'s account on the blockchain, and $\mathsf{E/D}$ is a symmetric encryption/decryption algorithm (e.g., AES). $\mathcal{U}$ randomly chooses $sk_{\mathcal{U}} \in Z_p^*$ as the secret key and computes $pk_{\mathcal{U}} = sk_{\mathcal{U}}P$ as the corresponding public key. $\mathcal{U}$ also chooses $k_{\mathcal{U}} \in Z_p$ as an encryption/decryption key for $\mathsf{E/D}$.

Outsource. $\mathcal{U}$ generates a new file and outsources it to $\mathcal{LS}$ as follows.

- $\mathcal{U}$ generates a new file F and encrypts it using $k_{\mathcal{U}}$ as:
$$C = \mathsf{E}(k_{\mathcal{U}}, F).$$
- Based on the current time, $\mathcal{U}$ acquires the hash values of φ-successive blocks that are latest confirmed on the Ethereum blockchain. The hash values of these blocks are denoted by $Bl_{t-\varphi+1}$, $Bl_{t-\varphi+2}$, $\cdots$, Bl_t, respectively, where t is the height of the block that is latest confirmed on the blockchain, and we recommend $\varphi \geq 12$ for Ethereum.
- $\mathcal{U}$ computes $\theta = H(C||Bl_{t-\varphi+1}||Bl_{t-\varphi+2}||\cdots||Bl_t)$ and $\sigma = sk_{\mathcal{U}}\theta$.
- $\mathcal{U}$ sends $\hat{C} = \{C, Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \cdots, Bl_t, \sigma\}$ to $\mathcal{LS}$.
- Upon receiving $\hat{C}$, $\mathcal{LS}$ checks whether the block corresponding to Bl_t is the latest one that is confirmed

6. We stress that an inherent requirement to design Chronos$^+$ is to support the case of multiple log servers, as discussed in Section 4.1.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

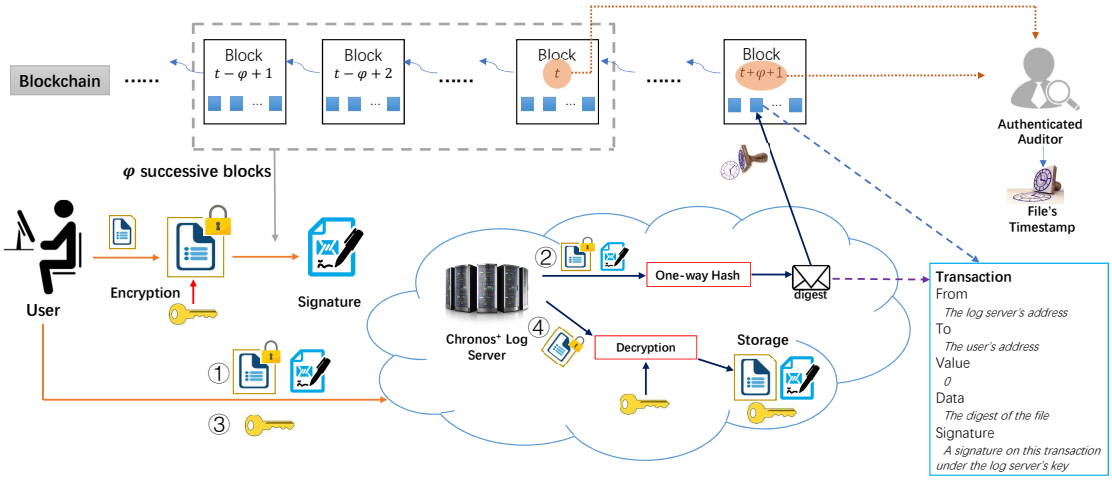


Fig. 5. The proposed Chronos+

on the blockchain[7]. If the checking fails, $\mathcal{LS}$ rejects it; otherwise, $\mathcal{LS}$ verifies the following equation:

$$e(\sigma, P) = e(H(C||Bl_{t-\varphi+1}||\cdots||Bl_t), pk_{\mathcal{U}}). \quad (1)$$

If Eq. 1 holds, $\mathcal{LS}$ accepts $\hat{C}$; otherwise, $\mathcal{LS}$ rejects it.

TimeStamp. $\mathcal{LS}$ time-stamps $\hat{C}$ as follows.

- $\mathcal{LS}$ computes a digest of $\hat{C}$ as
$$\delta = h(Bl_{t-\varphi+1}||Bl_{t-\varphi+2}||\cdots||Bl_t||C||\sigma).$$
- $\mathcal{LS}$ generates a transaction shown in Fig. 5, where $\mathcal{LS}$ transfers 0 Ether from its account $A_{\mathcal{LS}}$ to $\mathcal{U}$'s account $A_{\mathcal{U}}$ and sets δ on the data value of the transaction. $\mathcal{LS}$ then uploads the transaction to the Ethereum blockchain. Ideally, this transaction would be recorded in the block whose height is $t+\varphi+1$.
- Once the transaction is accepted and confirmed by the blockchain, $\mathcal{U}$ sends $k_{\mathcal{U}}$ to $\mathcal{LS}$.
- $\mathcal{LS}$ decrypts C by computing
$$F = \mathsf{D}(k_{\mathcal{U}}, C).$$
If the decryption fails, $\mathcal{LS}$ aborts; otherwise, $\mathcal{LS}$ stores
$$\{Bl_{t-\varphi+1}, \cdots, Bl_t, t+\varphi+1, F, k_{\mathcal{U}}, \sigma\}.$$

CheckStamp. Given $\{Bl_{t-\varphi+1}, \cdots, Bl_t, t+\varphi+1, F, k_{\mathcal{U}}, \sigma\}$, $\mathcal{A}$ can check the creation time of F as follows:

- $\mathcal{A}$ acquires the information of $A_{\mathcal{LS}}$ and $A_{\mathcal{U}}$ from the Ethereum blockchain.
- $\mathcal{A}$ extracts the number of transactions from $A_{\mathcal{LS}}$ to $A_{\mathcal{U}}$ based on the nonce value of $A_{\mathcal{LS}}$.
- $\mathcal{A}$ checks whether the number of transactions matches the number of data files generated by $\mathcal{U}$. If the checking fails, $\mathcal{A}$ aborts.
- $\mathcal{A}$ computes $C = \mathsf{E}(k_{\mathcal{U}}, F)$ and verifies the validity of σ. If the verification fails, $\mathcal{A}$ aborts.

7. In practice, the block corresponding to Bl_t may be not the latest one due to the delay caused by communication. However, this delay would not be long and the block can be accepted if it is one of the latest ones that is confirmed on the blockchain. For the sake of brevity, we do not consider the delay in this section.

- Based on the block height $t+\varphi+1$, $\mathcal{A}$ locates the block and extracts δ' from the corresponding transaction and verifies the following equation:
$$h(Bl_{t-\varphi+1}||Bl_{t-\varphi+2}||\cdots||Bl_t||C||\sigma) = \delta'.$$
If the verification fails, $\mathcal{A}$ aborts.
- $\mathcal{A}$ computes
$$ts_1 = \tau + \rho \cdot t, \quad (2)$$
$$ts_2 = \tau + \rho \cdot (t+\varphi+1), \quad (3)$$
where τ is 2015-07-30, 03:26:13 PM +UTC (i.e., the time the genesis block of Ethereum was appended) and ρ is the average time that a new block is mined from the day of 2015-07-30 to the day the block is appended to the blockchain in Ethereum. We will provide the details on setting ρ in Section 6.3.
- $\mathcal{A}$ outputs $[ts_1, ts_2]$ as a timestamp of F, which indicates that F was generated during $[ts_1, ts_2]$.

6.2 Support for batch time-stamping

In reality, multiple files from different users may be time-stamped concurrently. The individual time-stamping of these files for the log server and users could be tedious, inefficient, and inaccurate. Given n time-stamping tasks on n distinct files from n different file owners, it is more advantageous for the log server to batch these tasks together and time-stamp these files at one time. Therefore, we extend Chronos+ to support batch time-stamping, which is described below.

We assume that there are n users $\{\mathcal{U}_1, \mathcal{U}_2, \cdots, \mathcal{U}_n\}$.

Setup. The system parameters $\{p, P, G, G_T, e, h, H, A_{\mathcal{LCS}\_send}, A_{\mathcal{LCS}\_receive}, \mathsf{E/D}\}$ are determined with ℓ, where $A_{\mathcal{LCS}\_send}$ and $A_{\mathcal{LCS}\_receive}$ are two accounts of $\mathcal{LS}$ on the Ethereum blockchain, and other parameters are the same as the ones described before. For $i = 1, 2, \cdots, n$, $\mathcal{U}_i$ generates $\{sk_{\mathcal{U}_i}, pk_{\mathcal{U}_i}\}$ and $k_{\mathcal{U}_i}$ as the same as the basic scheme.

Outsource. For $i = 1, 2, \cdots, n$, $\mathcal{U}_i$ generates a new file and outsources it to $\mathcal{LS}$ as follows.

- $\mathcal{U}_i$ generates a new file F_i and encrypts it as
$$C_i = \mathsf{E}(k_{\mathcal{U}_i}, F).$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

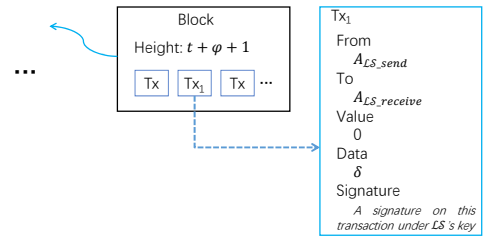


Fig. 6. Transaction on the Ethereum blockchain

- Based on the current time, $\mathcal{U}_i$ acquires $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \cdots, Bl_t$ from the Ethereum blockchain.
- $\mathcal{U}_i$ computes $\sigma_i = sk_{\mathcal{U}_i}\theta_i$, where
$$\theta_i = H(C_i||Bl_{t-\varphi+1}||Bl_{t-\varphi+2}||\cdots||Bl_t).$$
- $\mathcal{U}_i$ sends $\hat{C}_i = \{C_i, Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \cdots, Bl_t, \sigma_i\}$ to $\mathcal{LS}$.
- Upon receiving $\hat{C}_i$, $\mathcal{LS}$ checks the validity of $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \cdots, Bl_t$. In the batch time-stamping, n files from n users who choose the same $Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, \cdots, Bl_t$ are time-stamped simultaneously. Then $\mathcal{LS}$ computes $\theta_i = H(C_i||Bl_{t-\varphi+1}||Bl_{t-\varphi+2}||\cdots||Bl_t)$ and verifies the following equation:
$$e(\sum_{i=1}^{n}\sigma_i, P) = \prod_{i=1}^{n} e(\theta_i, pk_{\mathcal{U}_i}). \quad (4)$$
If the equation 4 holds, $\mathcal{LS}$ accepts $\hat{C}_i$.

TimeStamp. On receiving $\hat{C}_1$, $\hat{C}_2$, $\cdots$, $\hat{C}_n$, $\mathcal{LS}$ time-stamps them as follows.

- $\mathcal{LS}$ computes
$$\delta = h(Bl_{t-\varphi+1}||\cdots||Bl_t)||h(C_1||\sigma_1)||\cdots||h(C_n||\sigma_n).$$
- $\mathcal{LS}$ generates a transaction Tx_1 shown in Fig. 6, where 0 Ether is transferred from $A_{\mathcal{LS}\_send}$ to $A_{\mathcal{LS}\_receive}$, and δ is set to the data value of Tx_1.
- $\mathcal{LS}$ uploads the transaction to the Ethereum blockchain. Ideally, this transaction would be recorded in the block whose height is $t+\varphi+1$.
- Once the transaction is accepted and confirmed by the blockchain, $\mathcal{LS}$ sends the information of the corresponding block to all users.
- For $i = 1, 2, \cdots, n$, $\mathcal{U}_i$ verifies that $h(C_i||\sigma_i)$ has been recorded into the Ethereum blockchain. If the verification passes, $\mathcal{U}_i$ sends $k_{\mathcal{U}_i}$ to $\mathcal{LS}$.
- $\mathcal{LS}$ decrypts C_i by computing $F_i = \mathsf{D}(k_{\mathcal{U}_i}, C_i)$, and locally stores the data which is shown in Table. 1.

TABLE 1
Data stored on $\mathcal{LS}$

$Bl_{t-\varphi+1}, Bl_{t-\varphi+2}, ..., Bl_t$	$t+\varphi+1$	
F_1	$k_{\mathcal{U}_1}$	σ_1
$\cdots$	$\cdots$	$\cdots$
F_n	$k_{\mathcal{U}_n}$	σ_n

CheckStamp. This algorithm is the same as that in the basic scheme, we would not repeat it for the sake of brevity.

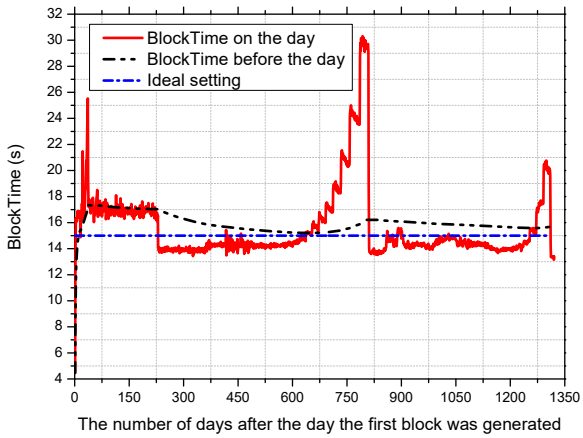


Fig. 7. BlockTime of Ethereum

6.3 Accuracy of height-derived timestamps

We denote the average time block mining on the Ethereum blockchain by BlockTime. Fig. 7 shows BlockTime[8], where the blue dash line indicates BlockTime of system setting (i.e., 15s), the red line indicates BlockTime on the x-th day after the day of τ (i.e., the genesis block was created, 2015-07-30 +UTC), and the black dot line indicates BlockTime from the day of τ to the x-th day after τ (x is the corresponding value of the X-axis). As shown in Fig 7, BlockTime in Ethereum is larger than the pre-set one, due to the network delay, the fluctuation of network hashing power and so on. Therefore, if we set $\rho = 15$ in equation 2 and 3, the height-derived timestamp is still not accurate. However, although BlockTime is not equal to 15 seconds, the chain growth property is not broken, since BlockTime in Ethereum still falls into a small range of time with respect to both short or long term, even if BlockTime is fluctuating on a single day. We assume that ρ_x is the average BlockTime from τ to the x-th day after the day of τ. In equation 2 and 3, ρ should be set to ρ_x when the block was appended to the blockchain on the x-th day after 2015-07-30 +UTC. We can compute ρ_x as
$$\rho_x = \frac{\sum_{j=1}^{x}\hat{\rho}_x}{x+1}, \quad (5)$$
where $\hat{\rho}_x$ denotes the BlockTime on the x-th day after the day of τ (shown by the red line in Fig. 7). We stress that BlockTime on each day is very important for the Ethereum. Multiple supernodes and full nodes have maintained and released Ethereum's BlockTime on each day in real time. Therefore, $\hat{\rho}_x$ can be easily derived from the Ethereum blockchain. To ensure the accuracy of height-derived timestamp, ρ_x should be periodically adjusted due to the fluctuation of BlockTime in Ethereum. For example, when $x = 1146$, i.e., the day is 2018-09-16, $\rho_x = 15.72$s. When $x = 1322$, i.e., the day is 2019-03-13, $\rho_x = 15.65704769$s.

Another factor that affects the accuracy of files' timestamps is the range of the time interval, i.e., the range of $[ts_1, ts_2]$ denoted by R_{TS}. In Chronos$^+$, for the x-th day after the day of τ, $R_{TS} = \varphi\hat{\rho}_x$. For the recommended $\varphi = 12$, R_{TS} in Chronos$^+$ is around 3 minutes. Although R_{TS} varies

8. The data are collected from Etherscan, https://etherscan.io

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

with BlockTime, it would not be too large in practice, due to chain growth property of the blockchain.

7 Security Analysis

In this section, security properties of Chronos$^+$ are analyzed from three aspects.

7.1 Resistance against malicious file owners

A malicious file owner may attempt to forward-date his file. Forward-dating files is a critical issue in the physical world, where the timestamp of a file could be a dated entry attached to the corresponding file recorded in a notebook, the adversary can attach a file to a dated entry that corresponds to a point in the future. However, it is hard to forward-date files in Chronos$^+$, due to the following two reasons.

First, the earliest creation time of the file in Chronos$^+$ is derived from the corresponding φ-successive blocks that are latest confirmed on the blockchain. Due to the property of (ι, φ)-chain quality, any adversary whose hashrate is less than 51% of the network's mining hashrate cannot fully control these φ-successive blocks [52]. Therefore, the hash value of these blocks cannot be pre-determined, which makes the pre-generation of the file's timestamp impossible. Second, the adversary cannot let miners in the blockchain system procrastinate on collecting the transaction containing the file's information to forward-date the file, since the security of blockchains ensures that a valid transaction could be recorded into the blockchain within a determined time interval [25], [46], [53].

The malicious file owner may also attempt to back-date his file. In this case, the goal of malicious file owner is to forge a timestamp for a newly created file to convince others that the file was created at a point in the past. The adversary has two possible strategies:

1) He integrates the file information into an existing transaction on the blockchain.
2) He creates a new transaction where the file information is integrated and records the transaction into a target block that has been chained to the blockchain at a point in the past.

We give a security analysis of Chronos$^+$, with respect to the two strategies above.

Strategy 1) requires the malicious file owner to modify the blockchain without detection. Specifically, the data field is a part of the transaction, and thereby is a part of the chained block. Adding a new data value to an existing transaction on blockchain essentially modifies the blockchain itself. This is impossible due to the φ-chain consistency which formalizes the security property that blockchain systems are inherently resistant to modification [24], [46].

Strategy 2) requires the malicious file owner to fork the blockchain. In particular, as shown in Fig. 8, in this attack, at a point in time, the blockchain has the form indicated by blue blocks in Fig. 8. The malicious file owner creates a new transaction where the information about the file is integrated, and attempts to record this transaction into a chained block on the blockchain (indicated by the first red block). Since each block includes the hash value of the previous one, if the malicious file owner substitutes the target block by the newly generated one (which contains the newly created transaction), the validity of the blockchain is broken. The only way to perform this attack is to fork the blockchain, i.e., he makes the blockchain including the newly generated block (shown within the yellow rectangle) become the main chain. However, due to the security of Ethereum, it is infeasible to fork the blockchain for adversaries whose budget is limited and hashrate is less than 51% of the network's hashrate.

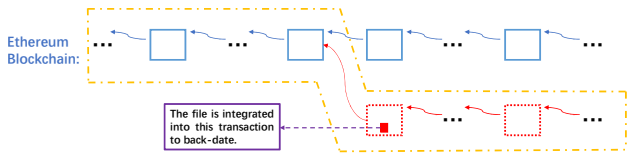


Fig. 8. Back-dating a file by forking the blockchain

7.2 Resistance against adversarial competitors

For an adversarial competitor in the system, we mainly focus on thwarting the attack of stealing the thunder (short for AST). In AST, the adversarial competitor first intercepts the file sent from a target user and changes the ownership of the file. Then, he requests a timestamp on the modified file from the log server. Note that AST can be performed by the adversarial competitor who compromises the log server.

Chronos$^+$ is secure against adversarial competitors, due to the proposed "unlock on delivery" paradigm. Particularly, in Chronos$^+$, the user requests the time-stamping service on the ciphertext from the log server. The log server integrates the digest of the ciphertext into a transaction on the blockchain to time-stamp the file. After the transaction is confirmed, the user then sends the encryption/decryption key to the log server. The log server decrypts the ciphertext and well maintains the file and the key. The authenticated auditor who can access the file and the key is able to check the correspondence between the digest of the ciphertext in the transaction and the file maintained by the log server. Since the file is encrypted when it is time-stamped, the malicious competitor cannot change the ownership of the file, even if he compromises the log server. Recall that the goal of the adversarial competitor is to steal the thunder, if he colludes with the log server to change the ownership of an existing file after the file is time-stamped, he cannot prove that the modified file was generated earlier than the existing one.

We further stress that the main goal of Chronos$^+$ is to provide a secure and accurate time-stamping service for users. We assume that the integrity of files stored on the log server has already been guaranteed by orthogonal techniques [9], [41], [54].

7.3 Resistance against compromised log server

In regards to the compromised log server, we prove that Chronos$^+$ is secure against equivocation attacks.

After a file is time-stamped, the log server may equivocate about the file on the blockchain to substitute the existing timestamp by a newly created one. This attack is shown in Fig. 9, where the log server creates a new

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

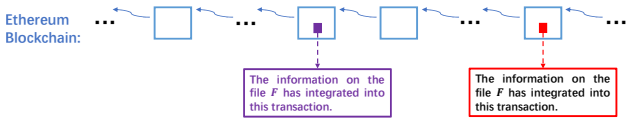


Fig. 9. Equivocation about the file F

transaction and integrates the information about a time-stamped file F into the transaction (since the log server has the encryption/decryption key, it can compute the ciphertext). Consequently, two transactions on the blockchain correspond to the same file, which causes an equivocation. The log server can only release the newly created transaction to show the timestamp of F.

However, in Chronos$^+$, such the attack can be easily detected. Note that when the authenticated auditor checks the timestamps of the files, it first checks whether the number of transactions the log server creates matches the number of files that have been time-stamped. If the checking fails, the timestamp of the file would be invalid. Since the log server's account used to create transaction is specially-crafted and dedicated, the number of transactions the log server creates can be easily obtained from the "nonce" field of the account. Therefore, Chronos$^+$ resists the compromised log server.

8 Performance Evaluation

In this section, the performance of Chronos$^+$ is evaluated and the practicality of Chronos$^+$ is measured. The experiment is conducted on a laptop with macOS, an Intel Core i7 CPU, and 16GB DDR3 of RAM. cryptographic operations are implemented by using C language and MIRACL library. The security level is chosen to be 80 bits for the evaluation, where the underlying elliptic curve is an MNT curve[9] whose base field size is 159 bits and embedding degree is 6. The feasibility of Chronos$^+$ for mobile users is also tested, where the experiment on the user side is conducted on a smartphone (HUAWEI MT2-L01) with the Android 4.2.2 system, a Kirin 910 CPU with memory 1250 MB, and we use SetCPU to change the frequency of the smartphone for the experiment.

8.1 Communication overhead

The communication overhead between the user and the log server is proportional to the size of the file to be time-stamped. In the following evaluation, we exclude the communication costs caused by sending/receiving the file. On the user side, the communication costs to time-stamp a file are $\varphi \cdot |\hat{h}| + \varphi \cdot |\hat{h}| + |\sigma| + |k_{\mathcal{U}}|$, where $|\hat{h}|$ is the size of a block's hash value in Ethereum, i.e., 256 bits. When $\varphi = 12$, the communication costs are around 0.8 KB.

On the log server side, the communication costs consist of three parts. The first one is to acquire hash values of φ-successive blocks from the Ethereum blockchain to verify the validity of the φ-successive blocks received from the user; The second one is to upload transactions to the Ethereum blockchain; The third one is to prove the timestamp of the file to the user and receive the decryption key from the user. In Chronos$^+$, the transaction is simple. The size of the transaction is about $109 + X$ bytes, where X is the size of the data value. In the single-user case, the communication costs on the log server are around 0.94 KB. Compared with the individual time-stamping, the batch time-stamping is more advantageous for the log server on the communication overhead when it serves multiple users simultaneously, which is shown in Fig. 10. Compared with the individual time-stamping, the batch time-stamping indeed reduces the log server's communication costs, as more than 40% of per-task communication costs are saved. The communication evaluation also demonstrates that the communication costs are acceptable in practice, since they can be negligible when being compared with the size of time-stamped files.

9. Actually, MNT curve is used to construct Type-3 pairings, i.e., $e : G_2 \times G_1 \rightarrow G_T$, where G_1 and G_2 are additive groups. Since Type-3 pairings are more efficient than Type-1 pairings when we choose the same security level, in the implementation of Chronos$^+$, we utilize Type-3 pairing to analyze the performance. This would not affect the feasibility and security of Chronos$^+$.

8.2 Computational overhead

The computational costs with respect to basic cryptographic operations are introduced in Table 2.

TABLE 2
Notation of operations

Symbol	Operation
$Hash_G$	hash a value into G
$Mult_G$	group operation in G
$Pair_{G_T}$	computing pairing $e(\chi, \varsigma)$ where $\chi, \varsigma \in G$
Enc	symmetric encryption/decryption operation
$Hash_{Z_p}$	hash a value into Z_p
Tx	conducting a transaction in Ethereum

On the user side, the data sent to the log server includes the encrypted file and the corresponding signature. Since generating files depends on the user's requirements, we would exclude this part of computation in the following. As such, the corresponding computation costs are

$$c \cdot Hash_G + c \cdot Mult_G + c \cdot Enc,$$

where c is the number of files to be time-stamped. We show the computation delay on the user in Fig. 11, where the experiments are conducted on a laptop and a smartphone with different frequencies (denoted by f). According to the experiment results, we can observe that the computation delay on the not-so-powerful smartphone is very short.

On the log server side, the computational costs are

$$2c \cdot Pair_{G_T} + c \cdot Hash_G + c \cdot Hash_{Z_p} + c \cdot Tx + c \cdot Enc,$$

where c is the number of files. In our experiment, the computational delay on the log server for a user is within 20ms. In reality, the log server would serve lots of users simultaneously. In this case, the batch time-stamping scheme would improve computation efficiency significantly. In Fig. 12, we show the experiment results of the comparison between the individual time-stamping and batch time-stamping with respect to the computational delay, which shows that the per-task computational costs can be reduced more than 45%. The experiment results prove that Chronos$^+$ is efficient and can be easily deployed in practice.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2947476, IEEE Transactions on Services Computing

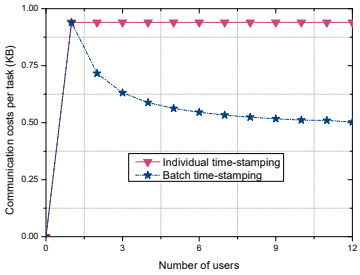


Fig. 10. Communication costs of individual and batch time-stamping

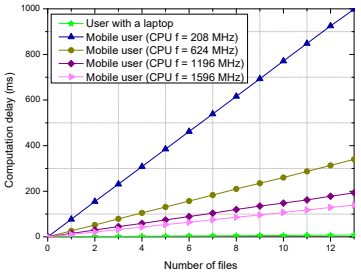


Fig. 11. Computation delay on the user w.r.t. the number of files to be time-stamped

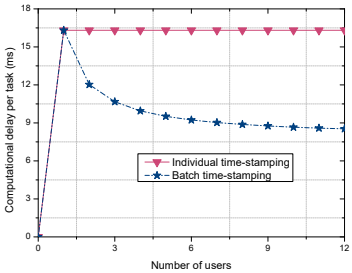


Fig. 12. Computation delay on the log server w.r.t. the number of users

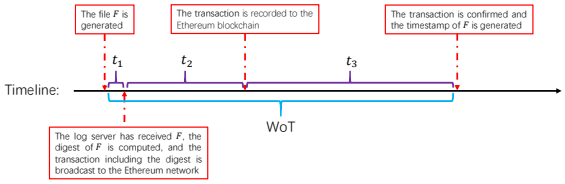


Fig. 13. Illustration of WoT

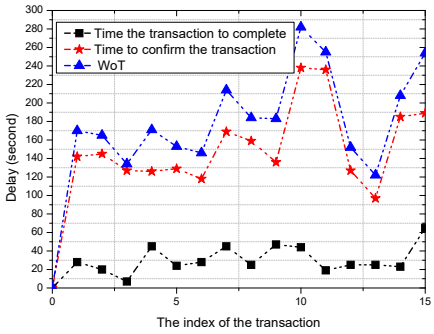


Fig. 14. Evaluation of WoT

8.3 Measurement of practicality

The practicality of Chronos$^+$ is measured in two aspects.

First, WoT of Chronos$^+$ is measured. WoT consists of three parts, as shown in Fig. 13. The first one is the time to process the file F, which has been evaluated in Section 8.2 and is denoted by t_1; The second one is the time to record the transaction to the blockchain (denoted by t_2); The third one is the time to confirm the transaction (denoted by t_3).

Since Chronos$^+$ is constructed on Ethereum, its WoT mainly depends on the time to record and confirm a transaction. Specifically, When the transaction including the digest of F is generated, it would be recorded to the blockchain, which requires a latency t_1. Then, the log server needs to confirm that the file is successfully time-stamped, which requires a latency t_2. In Ethereum, a block and its transactions are deemed to be confirmed if at least φ-successive blocks are mined following it. We evaluate t_2, t_3, and WoT by conducting 15 transactions on the Ethereum blockchain, where we use MyEtherWallet[10] as the wallet App. The evaluation results are shown in Fig. 14.

At the time of writing this paper (Mar. 2019), Chronos$^+$'s WoT is around 2.7 minutes. Since the Ethereum system would dynamically adjust the difficulty, generally, WoT of Chronos$^+$ would not exceed 300 seconds.

Second, the monetary costs to time-stamp a file is measured. Chronos$^+$ does not rely on smart contracts. The monetary costs to time-stamp files are mainly caused by conducting transactions on the blockchain and record the digest to the transaction. As of Mar. 2019, time-stamping a file requires one to pay about 5 US cents[11] for conducting a transaction (a hash value is included into the transaction) in Ethereum. This can be acceptable to users in respect of the value of the file protected by Chronos$^+$.

9 Conclusion

In this paper, potential threats towards outsourced time-sensitive files have been pointed out. Chronos$^+$, a secure blockchain-based time-stamping scheme for cloud storage systems has been proposed, where both the storage and time-stamping services are provided by a Chronos$^+$ log server which is subject to a cloud service provider. Chronos$^+$ provides an accurate way to prove that a file was created during a time interval formed by the earliest and latest creation times. The security of Chronos$^+$ has been analyzed, which has proven that Chronos$^+$ is secure against malicious users and compromised log server. A concept of window of time-stamping (WoT) has been introduced to measure the practicality of time-stamping schemes. A comprehensive evaluation has been conducted to demonstrate that Chronos$^+$ is efficient in terms of communication, computation, and monetary costs, and WoT.

For our future work, more complex functionalities in Chronos$^+$ need to be investigated. Specifically, in cloud storage systems, in addition to files, some operations (such as access to files and search for a file) on outsourced files performed by either cloud service providers or users should also be time-stamped for post investigations. This remains an open research issue that should be further explored.

Acknowledgement

This work is supported by the National Key R&D Program of China under Grant 2017YFB0802000, the National Natural Science Foundation of China under Grants 61872060, 61370203, the Sichuan Science and Technology Program under Grants 2019YFS0068, and the China Scholarship Council. The first author would like to thank the colleagues

10. https://www.myetherwallet.com/

11. https://bitinfocharts.com/comparison/ethereum-transactionfees.html

from BBCR lab, University of Waterloo, for their valuable discussions.

REFERENCES

[1] Y. Zhang, C. Xu, H. Li, H. Yang, and X. Shen, "Chronos: Secure and accurate time-stamping scheme for digital files via blockchain," in *Proc. ICC*, 2019, pp. 1–6.

[2] H. Wang, D. He, A. Fu, Q. Li, and Q. Wang, "Provable data possession with outsourced data transfer," *IEEE Trans. Services Computing*, to appear, doi: 10.1109/TSC.2019.2892095.

[3] Y. Yang, Z. Zheng, X. Niu, M. Tang, Y. Lu, and X. Liao, "A location-based factorization machine model for web service qos prediction," *IEEE Trans. Services Computing*, to appear, doi: 10.1109/TSC.2018.2876532.

[4] J. Liang, Z. Qin, S. Xiao, J. Zhang, H. Yin, and K. Li, "Privacy-preserving range query over multi-source electronic health records in public clouds," *Journal of Parallel and Distributed Computing*, accepted 2019, to appear.

[5] A. Zhou, S. Wang, Z. Zheng, C. Hsu, M. R. Lyu, and F. Yang, "On cloud service reliability enhancement with optimal resource usage," *IEEE Trans. Cloud Computing*, vol. 4, no. 4, pp. 452–466, 2016.

[6] M. Li, L. Zhu, and X. Lin, "Privacy-preserving traffic monitoring with false report filtering via fog-assisted vehicular crowdsensing," *IEEE Trans. Services Computing*, pp. 1–11, accepted 2019, to appear, doi: 10.1109/TSC.2019.2903060.

[7] Y. Zhang, C. Xu, H. Li, K. Yang, J. Zhou, and X. Lin, "HealthDep: An efficient and secure deduplication scheme for cloud-assisted ehealth systems," *IEEE Trans. Industrial Informatics*, vol. 14, no. 9, pp. 4101–4112, 2018.

[8] Y. Miao, X. Liu, K. R. Choo, R. H. Deng, J. Li, H. Li, and J. Ma, "Privacy-preserving attribute-based keyword search in shared multi-owner setting," *IEEE Trans. Dependable and Secure Computing*, pp. 1–15, accepted 2019, to appear, doi: 10.1109/TDSC.2019.2897675.

[9] A. Yang, J. Xu, J. Weng, J. Zhou, and D. S. Wong, "Lightweight and privacy-preserving delegatable proofs of storage with data dynamics in cloud storage," *IEEE Trans. Cloud Computing*, to appear, doi: 10.1109/TCC.2018.2851256.

[10] H. Li, Y. Yang, T. H. Luan, X. Liang, L. Zhou, and X. Shen, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Trans. Dependable and Secure Computing*, vol. 13, no. 3, pp. 312–325, 2016.

[11] X. Zhang, C. Xu, H. Wang, Y. Zhang, and S. Wang, "Fs-peks: Lattice-based forward secure public-key encryption with keyword search for cloud-assisted industrial internet of things," *IEEE Trans. Dependable and Secure Computing*, pp. 1–15, accepted 2019, to appear, doi: 10.1109/TDSC.2019.2914117.

[12] Y. Zhang, X. Lin, and C. Xu, "Blockchain-based secure data provenance for cloud storage," in *Proc. ICICS*, 2018, pp. 3–19.

[13] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Proc. CRYPTO*, 1990, pp. 437–455.

[14] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," in *Sequences II: Methods in Communication, Security, and Computer Science*, 1992, pp. 329–334.

[15] A. Buldas, H. Lipmaa, and B. Schoenmakers, "Optimally efficient accountable time-stamping," in *Proc. PKC*, 2000, pp. 293–305.

[16] H. Massias, X. S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirement," in *Proc. SITB*, 1999, pp. 1–8.

[17] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proc. USENIX ATC*, 2016, pp. 181–194.

[18] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *Proc. IEEE S & P*, 2017, pp. 393–409.

[19] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." https://bitcoin.org/bitcoin.pdf.

[20] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Survey & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[21] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[22] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou, "Detecting ponzi schemes on ethereum: Towards healthier blockchain technology," in *Proc. WWW*, 2018, pp. 1409–1418.

[23] A. Kiayias and G. Panagiotakos, "Speed-security tradeoffs in blockchain protocols," *IACR Cryptology ePrint Archive*, vol. 2015, pp. 1–19, 2015.

[24] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Proc. EUROCRYPT*, 2015, pp. 281–310.

[25] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Proc. CRYPTO*, 2017, pp. 324–356.

[26] H. Yang, Q. Zhou, M. Yao, R. Lu, H. Li, and X. Zhang, "A practical and compatible cryptographic solution to ADS-B security," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3322–3334, 2019.

[27] Y. Miao, Q. Tong, K. R. Choo, X. Liu, R. H. Deng, and H. Li, "Secure online/offline data sharing framework for cloud-assisted industrial internet of things," *IEEE Internet of Things Journal*, pp. 1–11, accepted 2019, to appear, doi: 10.1109/JIOT.2019.2923068.

[28] J. Ni, K. Zhang, X. Lin, and X. Shen, "Securing fog computing for internet of things applications: Challenges and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 601–628, 2018.

[29] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Trans. Dependable and Secure Computing*, pp. 1–13, accepted 2019, to appear, doi: 10.1109/TDSC.2019.2922958.

[30] H. Yang, X. Wang, C. Yang, X. Cong, and Y. Zhang, "Securing content-centric networks with content-based encryption," *Journal of Network and Computer Applications*, vol. 128, pp. 21–32, 2019.

[31] J. Weng, J. Weng, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," Cryptology ePrint Archive, report 2018/679, 2018.

[32] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Vehicular Technology*, vol. 68, no. 8, pp. 8050–8062, 2019.

[33] Z. Li, Z. Yang, and S. Xie, "Computing resource trading for edge-cloud-assisted internet of things," *IEEE Trans. Industrial Informatics*, vol. 15, no. 6, pp. 3661–3669, 2019.

[34] Y. Zhang, C. Xu, J. Ni, H. Li, and S. Xie, "Blockchain-assisted public-key encryption with keyword search against keyword guessing attacks for cloud storage," *IEEE Trans. Cloud Computing*, pp. 1–14, accepted 2019, to appear, doi: 10.1109/TCC.2019.2923222.

[35] D. Liu, A. Alahmadi, J. Ni, X. Lin, and X. Shen, "Anonymous reputation system for iiot-enabled retail marketing atop pos blockchain," *IEEE Trans. Industrial Informatics*, vol. 15, no. 6, pp. 3527–3537, 2019.

[36] J. Coleman, "Universal hash time," https://www.youtube.com/watch?v=phXohYF0xGo.

[37] E. Landerreche, C. Schaffner, and M. Stevens, "Cryptographic timestamping through sequential work," CWI Amsterdam, Tech. Rep., 2018.

[38] W. Jiang, H. Li, G. Xu, M. Wen, G. Dong, and X. Lin, "Ptas: Privacy-preserving thin-client authentication scheme in blockchain-based pki," *Future Generation Computer Systems*, vol. 96, pp. 185–195, 2019.

[39] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J. Liu, Y. Xiang, and R. H. Deng, "CrowdBC: A blockchain-based decentralized framework for crowdsourcing," *IEEE Trans. Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2018.

[40] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Proc. ASIACRYPT*, 2001, pp. 514–532.

[41] Y. Zhang, C. Xu, X. Liang, H. Li, Y. Mu, and X. Zhang, "Efficient public verification of data integrity for cloud storage systems from indistinguishability obfuscation," *IEEE Trans. Information Forensics and Security*, vol. 12, no. 3, pp. 676–688, 2017.

[42] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE BigData Congress*, 2017, pp. 557–564.

[43] ——, "Blockchain challenges and opportunities: A survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, 2018.

[44] M. Conti, S. Kumar, C. Lal, and S. Ruj, "A survey on security and privacy issues of bitcoin," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3416–3452, 2018.

[45] C. Xu, K. Wang, P. Li, S. Guo, J. Luo, B. Ye, and M. Guo, "Making big data open in edges: A resource-efficient blockchain-based

approach," *IEEE Trans. Parallel and Distributed Systems*, vol. 30, no. 4, pp. 870–882, 2018.
[46] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. CRYPTO*, 2017, pp. 357–388.
[47] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Proc. CRYPTO*, 2015, pp. 585–605.
[48] Y. Zhang, C. Xu, X. Lin, and X. Shen, "Blockchain-based public integrity verification for cloud storage against procrastinating auditors," *IEEE Trans. Cloud Computing*, pp. 1–15, accepted 2019, to appear, doi: 10.1109/TCC.2019.2908400.
[49] K. Yang, Z. Liu, X. Jia, and X. Shen, "Time-domain attribute-based access control for cloud-based video content sharing: A cryptographic approach," *IEEE Trans. Multimedia*, vol. 18, no. 5, pp. 940–950, 2016.
[50] H. Ren, H. Li, Y. Dai, K. Yang, and X. Lin, "Querying in internet of things with privacy preserving: Challenges, solutions and opportunities," *IEEE Network*, vol. 32, no. 6, pp. 144–151, 2018.
[51] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Trans. Information Forensics and Security*, vol. 14, no. 4, pp. 870–885, 2018.
[52] C. Pierrot and B. Wesolowski, "Malleability of the blockchain's entropy," *Cryptography and Communications*, vol. 10, no. 1, pp. 211–233, 2018.
[53] R. Goyal and V. Goyal, "Overcoming cryptographic impossibility results using blockchains," in *Proc. TCC*, 2017, pp. 529–561.
[54] X. Zhang, H. Wang, and C. Xu, "Identity-based key-exposure resilient cloud storage public auditing scheme from lattices," *Information Sciences*, vol. 472, pp. 223–234, 2018.

Yuan Zhang (S'16) received his B.E. degree in University of Electronic Science Technology of China (UESTC), China, in 2013. He is currently a Ph.D. candidate at the School of Computer Science and Engineering, University of Electronic Science Technology of China, and is a visiting Ph.D. student at BBCR Lab, Department of Electrical and Computer Engineering, University of Waterloo, Canada. His research interests include applied cryptography, data security, and blockchain technology. He is a student member of IEEE.

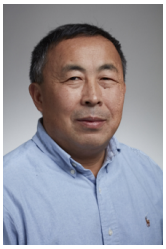
Chunxiang Xu (M'06) received the B.Sc. and M.Sc. degrees in applied mathematics from Xidian University, Xi'an, China, in 1985 and 2004, respectively, and the Ph.D. degree in cryptography from Xidian University in 2004.

She is currently a Professor with the Center for Cyber Security, the School of Computer Science and Engineering, UESTC. Her research interests include information security, cloud computing security and cryptography. She is a member of IEEE.

Nan Cheng (S'12–M'16) received the B.E. and M.S. degrees from Tongji University, Shanghai, China, in 2009 and 2012, respectively, and the Ph.D. degree from the University of Waterloo, Waterloo, ON, Canada, in 2016. He is currently a Professor with School of Telecommunication, Xidian University. His current research focuses on big data in vehicular networks and self-driving system.

Hongwei Li (M'12–SM'18) is currently the Head and a Professor at Department of Information Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China. He received the Ph.D. degree from University of Electronic Science and Technology of China in June 2008. He worked as a Postdoctoral Fellow at the University of Waterloo from October 2011 to October 2012. His research interests include network security and applied cryptography. He is the Senior Member of IEEE, the Distinguished Lecturer of IEEE Vehicular Technology Society.

Haomiao Yang (M'12) received the M.S. and Ph.D. degrees in computer applied technology from University of Electronic Science and Technology of China, Chengdu, China, in 2004 and 2008, respectively. He has worked as a Post-Doctoral Fellow in the Research Center of Information Cross over Security at Kyungil University. His research interests include cryptography, cloud security, and the cyber security for aviation communication. Now, he is an associate professor in School of Computer Science and Engineering and Center for Cyber Security of UESTC.



Xuemin (Sherman) Shen (M'97–SM'02–F'09) received the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 1990. He is currently a University Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is a registered Professional Engineer of Ontario, Canada, an Engineering Institute of Canada Fellow, a Canadian Academy of Engineering Fellow, a Royal Society of Canada Fellow, and a Distinguished Lecturer of the IEEE Vehicular Technology Society and Communications Society.

Dr. Shen received the R.A. Fessenden Award in 2019 from IEEE, Canada, the James Evans Avant Garde Award in 2018 from the IEEE Vehicular Technology Society, the Joseph LoCicero Award in 2015 and the Education Award in 2017 from the IEEE Communications Society. He has also received the Excellent Graduate Supervision Award in 2006 and the Outstanding Performance Award 5 times from the University of Waterloo and the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada. He served as the Technical Program Committee Chair/Co-Chair for the IEEE Globecom'16, the IEEE Infocom'14, the IEEE VTC'10 Fall, the IEEE Globecom'07, the Symposia Chair for the IEEE ICC'10, the Tutorial Chair for the IEEE VTC'11 Spring, the Chair for the IEEE Communications Society Technical Committee on Wireless Communications, and P2P Communications and Networking. He is the Editor-in-Chief of the IEEE INTERNET OF THINGS JOURNAL and the Vice President on Publications of the IEEE Communications Society.