IoETTS: A Decentralized Blockchain-based Trusted Time-stamping Scheme for Internet of Energy

Bin Qian[1,2], *Yi Luo*[1,2*], *Jiaxiang Ou*[3], *Yong Xiao*[1,2], *Houpeng Hu*[3]

[1] *Institute of Metrology Technology, Electric Power Research Institute, China*
[2] *Guangdong Provincial Key Laboratory of Intelligent Measurement and Advanced Metering of Power Grid, China*
[3] *Guizhou Power Grid Co., LTD, China*
qianbin@csg.cn, luoyi_csg@outlook.com, oujx@gzsyy.csg.cn, xiaoyong@csg.cn, huhp@gzsyy.csg.cn

Abstract

As a new-style smart grid, Internet of Energy (IoE) is important and how to provide its trusted time-stamping service becomes a hit. For example, an energy provider needs to prove he/she transferred some energy to a consumer at some time. Nevertheless, traditional trusted time-stamping scheme with a central service provider is not suitable for IoE. Some researchers try to solve this problem via blockchain, due to its decentralization, traceability and tamper-proof. However, there are still challenges when using blockchain. Some have to introduce another kind of central participant. Some have to face the problem of accuracy and availability when using the Bitcoin blockchain. Some have to generate too many extra transactions. To address the aforementioned problems, we propose a fully decentralized trusted time-stamping scheme without any central participant and fulfill six design goals. Compared with the state-of-the-art blockchain-based time-stamping scheme named Chronos, our scheme enjoys less cryptographic operations. We then tested our scheme in the development (local) network and two live networks of the Ethereum. The experiment shows that we have implemented a simple, effective, accurate and low-cost decentralized trusted time-stamping scheme.

Keywords: Trusted time-stamping, Smart grid, Internet of energy, Blockchain

1 Introduction

Due to the rising price of energy and the negative environmental impacts of fossil fuels, many countries are trying to introduce distributed energy such as renewable energy to build modern power systems [1]. This kind of energy system is usually called the Internet of Energy (IoE) or smart grid 2.0 [2]. The IoE is an extended concept of the smart grid. While smart grids make advances in sensing, communication and control, the new concept IoE is thought to be an internet-style way for energy issues. There are some key features in such a system [3]: (1) Mutual untrusted participants, such as large-scale distributed energy generation systems, storage systems and consumers must interact with each other for secure and reliable delivery of energy. (2) Energy and information are exchanged between a wide variety of participants via the internet. (3) Allowing peer-to-peer energy exchanges, the IoE should provide a new business model for a more open market. For example, it may provide peer-to-peer billing between the energy generator and consumer.

Consequently, the decentralized untrusted participants, the information flow on the internet and the open business market flourish a new research challenge called Forensic Science [4]. The essence is to provide evidence for the activities that occur in this kind of system. For instance, an energy generator may have to prove he/she transferred some energy to a consumer at some time for a reasonable income. Evidence should be agreed by different participants unanimously to execute a penalty when electricity theft activities occur. Indeed, there is a basic service called secure digital timestamps or Trusted time-stamping [5] which addresses these needs. A trusted timestamp is used to prove the existence of certain data before a certain point without the possibility that the owner can backdate or postdate it. Thus trusted time-stamping can become an approach in the IoE to prove an event occurs among the participants. However, traditional trusted time-stamping schemes like the RFC 3161 standard [6] and the Digital Time Stamp [7] usually need a Trusted Third Party (TTP) acting as a time-stamping authority (TSA). Because of the decentralized untrusted participants in the IoE, a centralized solution may not fit this scenario well. Then, in 2015 a decentralized time-stamping scheme [8] was proposed by using the famous cryptocurrency Bitcoin [9]. Bitcoin is a peer-to-peer electronic cash system and its underlying technology called blockchain has become a hit due to its key features of decentralization, traceability and tamper-proof.

Since then, finding a decentralized time-stamping solution via blockchain has drawn great interest from researchers. Thomas Hepp et al. proposed a blockchain-backed system called OriginStamp [10]. They gave insights into the implementation of a decentralized timestamp and presented a time-stamping approach also using the Bitcoin blockchain. The difference is the core of their scheme is implemented independently of the Bitcoin protocol to avoid the scaling problem [11]. In 2019, Yuan Zhang et al. [12] used the Ethereum blockchain to realize a trusted time-stamping service called Chronos for a long delay may occur when the systems are designed via Bitcoin. In order to let a

*Corresponding Author: Yi Luo; E-mail: luoyi_csg@outlook.com
DOI: 10.53106/160792642023032402027

file's timestamp be formed by a more accurate time, Chronos proves the existence of a file corresponding to a time interval by embedding the information of a sufficient number of consecutive confirmed blocks into the file. In 2021, Gabriel Estevam et al. [13] showed that some trusted time-stamping schemes given by sending transactions containing the time-stamped data to the blockchain might not be accurate. They used smart contracts on the Ethereum blockchain instead of the block-based timestamps and took the cost of the time-stamping service into account.

Generally, all these techniques have given a kind of decentralized time-stamping solution via blockchain and may fit different use cases with distinguishing features. Nevertheless, there are still limitations when they are used in the IoE system: Firstly, a well-designed solution should prevent malicious users from tampering with a timestamp, forging a proved timestamp, or denying a time-stamping service. However, some of them are constructed on Bitcoin and may suffer from a long delay and up to two-hour errors [12]. Secondly, some of these techniques still have a service provider which seems to be a kind of trusted third party. Because the IoE includes a variety of users who have different interests, a fully decentralized scheme is needed. An ideal time-stamping solution should not involve any kind of the third party. Finally, as a basic service, the trusted time-stamping service should be as simple as possible and easy to integrate into the IoE. Many of the solutions are trying to pack the record into a block directly. These block-based timestamps will increase the complexity of the IoE system and are hard to be used in a real system. To eliminate these limitations, we try to propose a novel IoE trusted time-stamping solution that is fully decentralized, secure, robust, and easy to invoke by the IoE system. Specifically, our contributions are summarized as follows:

1) We propose a blockchain-based decentralized time-statmping scheme for internet of energy. To the best of our knowledge, our IoE trusted time-stamping called IoETTS is the first work to take the key features of the IoE into account and eliminate any kind of trusted third party (TTP).

2) We address six design goals according to the above discussion and provide two threat models for the decentralized time-stamping scheme. Theoretical analysis is given to show that our proposed scheme fulfills the design goals and approaches to resist all the attacks in the threat models.

3) Compared with the state-of-the-art blockchain-based time-stamping scheme named Chronos, our scheme enjoys less cryptographic operations, thereafter we conduct experiments both in the development network and testnet of the Ethereum blockchain and the experiment shows that we have implemented a simple, effective, accurate and low-cost decentralized trusted time-stamping scheme.

The rest of this paper is arranged as follows. In Section 2 we give the related work of time-stamping service. In Section 3 we introduce the necessary preliminary knowledge. In Section 4 and 5 we give our system model and security analysis. In Section 6 we provide an evaluation in the development (local) network and testnet (Ropsten and Rinkeby networks) of the Ethereum blockchain. Finally, in Section 7 we draw a brief conclusion.

2 Related Work

In this section, we review the related works on the techniques which provide trusted time-stamping services.

The time-stamping technique was first proposed by Haber et al. [14] for addressing the problem of time-stamping digital files. In the scheme, a hashchain is used to link a sequence of digital files by a cryptographic hash function and a Trusted Third Party (TTP) is needed to give the time-stamping service. There is an assumption that the time-stamping request sequence cannot be known in advance and a new chain should be reformed if an adversary compromises the TTP. Following this scheme, [15] and [16] tried to provide a more efficient technique. Later, a trusted time-stamping service is standardized in RFC 3161 [6] as a basic service.

However, the techniques mentioned above are not fit IoE very well since they all need a Trusted Third Party (TTP). An IoE system usually contains mutual untrusted participants and provides a more open market. A decentralized trusted time-stamping service is needed in this kind of system. It means we should find a TTP-free time-stamping service for IoE. Since blockchain has become a powerful technology with characters of decentralization, tamper-proof and traceability, several techniques were proposed to give a decentralized service based on blockchain [17]. [8, 10] present their trusted time-stamping schemes which are based on the decentralized Bitcoin blockchain to store anonymous, tamper-proof timestamps for digital content. They showed that cryptocurrencies can be used as a decentralized trusted time-stamping ledger. However, a transaction cannot be confirmed by the Bitcoin blockchain network immediately for the limit of the Bitcoin [18]. In [19], Wang et al. proposed a repute-based consensus protocol for blockchain-enabled IoT systems to reach a consensus rapidly and safely. [20] tried to design more application protocols (i.e., key agreement) based on blockchain. Except for logging transactions into the blockchain directly, the Ethereum blockchain also provides a more high-level way by smart contracts [21]. And [13] created trusted timestamps by using smart contracts on the Ethereum blockchain with higher accuracy of milliseconds. But they still do not fit IoE very well because they may increase the complexity of the system or introduce a service provider which seems to be a kind of trusted third party.

3 Preliminaries

3.1 Trusted Time-Stamping Service

One of the current challenges in the IoE is creating a verifiable timestamp. Activities that occur in the system should be accepted by mutual untrusted participants. For example, an energy provider needs to prove he/she transferred some energy to a consumer at some time. The energy exchange activities will be recorded by sensors or smart meters in the IoE. And these records rely on accurate clocks on devices. Incorrectly times-tamped records may mislead the participants in the IoE and further challenge the security of the system. A trusted time-stamping service should contain two basic functionalities [22].

• **Creating a timestamp**: Any systems and devices in the IoE could use the trusted time-stamping service to create a timestamp for a record. Anyone could use this for auditing and supervising purposes. Businesses and individuals would no longer be able to backdate or postdate the records, whether relating to taxes or other duties, allowing the supervisors to ensure they are created on time without the need to collect them. It allows preventing records from being hidden, withheld or destroyed without detection.

• **Verifying a timestamp**: The ability to independently verify timestamps could also be highly desirable in the IoE, where each participant may require a confirmation from its own business activities. By using the trusted time-stamping service, a single timestamp may be verifiable and accepted by all participants without the need to trust any third parties.

Accordingly, we formally define four algorithms for a trusted time-stamping scheme: **InitialParams**, **AddStamp**, **UnlockStamp** and **VerifyStamp**.

• **InitialParams**: On inputting a user's account identity accountld, the algorithm generates a corresponding pair of keys (*PK*, *SK*) and then make the tuple (accountld, *PK*, *SK*) public.

• **AddStamp**: On inputting a user's record for certain behavior and its secret key (Record, SK), the algorithm creates a time stamp for the current user and output the tuple (*H*, *C*, Sig). *H* is a one-way hash of the Record. *C* is the record encrypted by an one-time secret key *sk*. And *Sig* is the signature on H by the asymmetric key *SK*.

• **UnlockStamp**: On inputting a user's one-time secret key and one of its timestamp (*sk*, *H*), the algorithm can reveal a time stamp. Note that the user's one-time secret key *sk* is logged into the corresponding record and then can be used to decrypt the encrypted record.

• **VerifyStamp**: On inputting a user's time stamp, the record for certain behaviour and its public key (*Record*, *H*, *PK*), the algorithm outputs True if the time stamp is valid. otherwise, it outputs False.

3.2 Cryptographic Functions

Cryptography is an indispensable tool used to protect the information in computing systems. In our scheme, we also used some cryptographic functions to provide data confidentiality, data integrity and data deniability. All these make the trusted time-stamping service suffice to the design goals in Section 3.5. Cryptographic functions are used everywhere and by billions of people worldwide on a daily basis. They are used to protect data at rest and data in motion. There are different schemes or functions for different applications. Here we introduce those we used in our scheme.

• **Collision resistant hashing**: In our scheme, we use SHA256 [23] which is a function that hashes long messages into 256-bit digests. It is believed that finding collisions for SHA256 is difficult.

• **Digital Signatures**: In our scheme, we use the Elliptic Curve Digital Signature Algorithm (ECDSA) [24] as a digital signature scheme which uses elliptic curve cryptography.

• **Encryption**: In our scheme, we use the Advanced Encryption Standard (AES) [25] as an encryption scheme. The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

3.3 Blockchain

The concept of Blockchain was originated from the famous cryptocurrency Bitcoin [1]. In the past decade, we have witnessed a rapid evolution in blockchain technologies due to its characteristics of decentralization, immutability, and self-organization.

• **Classification**: There are four main types of blockchain networks: *public blockchains*, *private blockchains*, *consortium blockchains* and *hybrid blockchains*. Different use cases require different types of blockchain. *Public blockchain* is non-restrictive and permissionless, anyone with internet access can sign on to a blockchain platform to become an authorized node. This user can access current and past records and conduct mining activities, the complex computations used to verify transactions and add them to the ledger. *Private blockchain* works in a restrictive environment like a closed network, or that is under the control of a single entity. Instead of just anyone being able to join and provide computing power, private blockchains typically are operated as a small network inside a company or organization. *Hybrid blockchain* lets organizations set up a private, permission-based system alongside a public permissionless system, allowing them to control who can access specific data stored in the blockchain, and what data will be opened up publicly. While *Consortium blockchain* also known as a federated blockchain, is similar to a hybrid blockchain in that it has private and public blockchain features. But it's different in that multiple organizational members collaborate on a decentralized network.

• **Smart Contract**: Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. Smart contracts are a type of Ethereum account. This means they have a balance, and they can send transactions over the network. However, they're not controlled by a user, instead they are deployed to the network and run as programmed. User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code. Smart contracts cannot be deleted by default, and interactions with them are irreversible. The core function is written in a smart contract in our scheme to provide the trusted time-stamping service.

3.4 Threat Models

Here we present two threat models specially for a trusted timestamp service.

1) **Record Owner Model**: Business activities that occurred in IoE are thought to be arguable and untrustable. And users who are related to a specific business sometimes need to stamp a record of activity. For example, an energy provider needs to prove he/she transferred some energy to a consumer at some time. The energy provider will use the trusted time-statmping service to set a record on the blockchain. Everyone can verify the validation of this record. Here the energy provider who used the trusted service to stamp a record is called the record owner. We assume that a

record owner is a rational one. It means he/she will generate a valid time-statmping record in order to pass the verification later and get some profits. For example, the consumer will pay the bill only after he/she verifies that the energy exchange activity record is valid and time-stamped correctly. However, a record owner may attempt to backdate or postdate existing records to increase his profits after he/she gets paid by the consumer. For example, a malicious record creator may attempt to tamper with existing timestamps of records to avoid his/her taxes. We stress that the record owner would not be misbehaviour when she/he requests the timestamp of a record since an invalid record will cause charge failure.

2) **Adversary Model**: An adversary is a valid user in the system. He/She may attempt to forge a record time-stamped earlier than the record created by the real record owner yet with the same content as the original one. Still, we take the energy exchange business as an instance. When a user (record owner) uploads a record to be time-stamped, a malicious user (adversary) can stop the record from being stamped on the blockchain for a while and forge the record to change the ownership and timestamp of the record. An adversary can get paid by the consumer after doing this. Actually, there is a possibility for an adversary to forge a record and timestamp it earlier. Since trusted time-stamping services provided by a smart contract are actually composed of transactions executed and confirmed by the blockchain, a miner/node in the blockchain can get the data of a transaction before it is confirmed by the blockchain. Therefore, any miner/node can be considered as an adversary.

3.5 Design Goals

In order to ensure the sustainability and resiliency of the ecosystem, a trusted time-stamping service in an IoE must fulfil numerous requirements. Thus, here we describe the main design goals, and we introduce the criteria needed to evaluate the suitability for IoE use cases.

Availability: the availability implies that services must be accessible to legitimate users on demand. Thus, a system must be resilient against denial of service attacks especially those who target the time-stamping service. The scheme should prove that a record or data 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.

Efficiency: The time-stamping service should not introduce heavy computation and communication costs on both the blockchain system and users in IoE. The blockchain system should be able to handle multiple tasks from different users simultaneously. The time interval for a timestamp to be securely recorded should be as short as possible.

Integrity: Maintaining integrity is the crucial requirement that each record to be proved must ensure. In our context, integrity is a record or data integrity which can be divided into two parts: 1) Transactions integrity: an exchanged record must not be altered or modified when it is sent to be written on the blockchain by the smart contract. 2) Storage integrity: a stamped record on the blockchain must not be altered or modified.

Scalability: In our context, scalability represents the ability to ensure that the system size has no impact on its performance. For example, if the number of the used things explodes, the time needed for a time stamping service, must not be affected.

Undeniability: It refers to the ability to ensure that an entity cannot deny having performed a given action, e.g. a device cannot deny having stamped a record.

Identification: The identification represents the main requirement in the majority of IoE use cases. It represents the contrary of anonymity which ensures that any entity can make use of the system all within ensuring being anonymous to all systems entities. For example, in energy monitoring where a sensor monitors the level of a district. When this sensor sends information to the monitoring platform, the latter must know exactly which sensor is communicating in order to decide about the actions to provide.

4 Proposed Scheme

4.1 System Overview

In this subsection, we will describe a general time-stamping scheme to provide decentralized trusted time-stamping service via blockchain.

As is shown in Figure 1, the time-stamping scheme is consisted with two processes: creating a timestamp and verifying a timestamp.

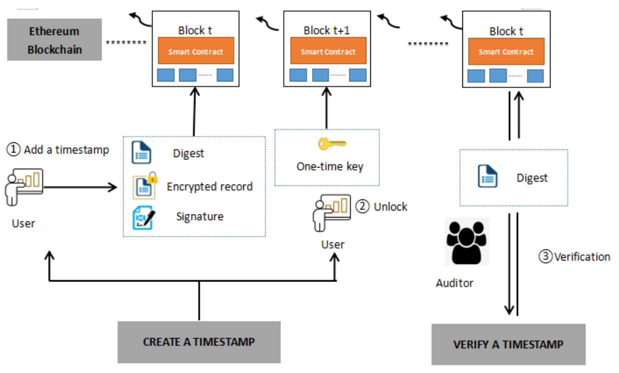


Figure 1. System overview

1) Creating a timestamp

In this process, a user generates his identity key pair (*PK*/*SK*) and a single-use secret key *sk*. The key pair (*PK*/*SK*) is used to generate a digital signature of a record to be timestamped. And the secret key sk is used to encrypt the record. The key pair (*PK*/*SK*) will not change/update frequently whilst the secret key *sk* will regenerate when someone tries to create a new timestamp. To create a timestamp for a record, a user should follow two steps: Add a timestamp: Firstly, a user calculates the hash value of the record to be timestamped *H*(*Record*). Then he/she generate a digital signature using createSign (*H*, *SK*) with the asymmetric secret key *SK* and encrypt the record (*Record*, *sk*) with the single-use symmetric key *sk*. Finally, he/she can invoke the smart contract to add a timestamp into the Ethereum blockchain with a Web3 smart contract we provided **AddS**(*H*, *C*, *Sig*). Unlock a timestamp: Once the operation **AddS**(*H*, *C*, *Sig*) is confirmed by the Ethereum blockchain, the user publishes the secret key *sk* into the blockchain by the operation **UnlockS**(*H*, *sk*). After this, anyone who tries to verify this timestamp can decrypt the encrypted record in the first step with the secret key *sk*. Note that the information of a time-stamped record is stored as a key-value map table in the blockchain where the key of the map is the digest of the record. We will give more details in the Section 4.3.

2) Verifying a timestamp

Anyone can verify a time-stamped record in the blockchain. For instance, when an energy provider has created a timestamp for a record about transferring some energy to a consumer, he/she will give the digest of this record to the consumer to verify. If the timestamp is valid, the consumer will pay the bill. Here the consumer plays as an auditor in Figure 1.

When an auditor wants to verify a time-stamped record, he/she firstly invokes the smart contract to get the information of the record by **CheckS**(*H*). The operation **CheckS**(*H*) will return a tuple value (*R*, *Sig*, *C*, *T*, *sk*). Where *R* is an error code where a non-zero value indicates there is some kind of errors that happened such as the record doesn't exist. *Sig* is the digital signature of the record. *C* is the ciphertext of the record. *T* is the block time when the record is time-stamped. *sk* is the symmetric key to encrypt/decrypt the record. The value of (*sig*, *C*, *T*, *sk*) will be none if errors are found. After getting the tuple value (*R*, *Sig*, *C*, *T*, *sk*), the auditor can decrypt the ciphertext by D(*C*, *sk*) and see the plaintext of the record. Finally, he/she can verify the validity of the time-stamped record by *V*(*PK*, *Sig*, *H*(*D*)). The operation *V*(*PK*, *Sig*, *H*(*D*)) will calculate verify(*PK*, *Sig*) to get a *H*(*D*)' and then compare it with *H*(*D*). The time-stamped record is valid if *H*(*D*)' equals *H*(*D*).

4.2 Algorithms Design

In this subsection, we will describe the design of the four algorithms defined in Section 3.1 in detail. In order to make the trusted time-stamping service as simple as possible to be integrated into an IoE system, we assume anyone who tries to use the trusted time-stamping service can ignore the underlying design in smart contracts of the blockchain. The details are as follows:

InitialParams: Anyone who wants to use the trusted time-stamping service should have a pair of asymmetric secret key (*PK*, *SK*). As is shown in Algorithm 1, we provide the function InitialParams for a user to generate a pair of keys and then make his/her *PK* public. The method *GenerateKeyPair* is an interface of an open source javascript cryptographic library to generate keys. We use the Elliptic Curves Digital Signature Algorithm (ECDSA) on the "secp256k1" [26] curve.

Algorithm 1. InitialParams

```
Input (accountld)
Output (PK, SK)
  1: (PK, SK) =
  GenerateKeyPair('eccurve', 'secp256k1')
  2: Publish (accountId, PK);
  3: Return (PK, SK)
```

AddStamp: As is shown in Algorithm 2, we provide the function AddStamp for a user to create a time stamp for a record. Firstly, we generate a single-use secret key "*sk*" and encrypt the record with Advanced Encryption Standard (AES) on "aes-256-cbc". The method *Encrypt* is an interface of an open source javascript cryptographic library to encrypt data. The parameter "*iv*" is an initialization vector used to ensure that the same value encrypted multiple times, even with the same secret key, will not always result in the same encrypted value. If you use each key only a single time, not using an "*iv*" is fine. Then we calculate the hash of the record and create the digital signature of it. Finally, we invoke an interface of the smart contract to add (*H*, *C*, *Sig*) into the blockchain. Note that in this algorithm, the final function **AddS** in the last line is a smart contract function which will be described in the next subsection.

Algorithm 2. AddStamp

```
Input (Record, SK)
  1: sk = randomBytes(32);
  2: iv = randomBytes(16);
  3: C = Encrypt(aes-256-cbc, Record, sk || iv);
  4: H = SHA256(Record);
  5: Sig = ECDSA.createSign(H, SK);
  6: await AddS(H, C, Sig);
```

UnlockStamp: As is shown in Algorithm 3, this part is just to patch a *sk* into the timestamp in the blockchain. Actually, we divided the process of creating a timestamp into two parts AddStamp and UnlockStamp. We make sure that the plaintext of the record is unknown except the owner until the information (*H*, *C*, *Sig*) is confirmed by the blockchain. The reason for this design will give in Section 5. Also note that in this algorithm, the function **UnlockS** is a smart contract function which will be described in the next subsection.

Algorithm 3. UnlockStamp

```
Input (sk, H)
  1: await UnlockS(H, sk);
  2: Return
```

VerifyStamp: An auditor who wants to verify a time- stamped record can use this function. The owner should provide the original record and his/her public key (*Record*, *PK*). Firstly, we invoke the smart contract to get the information of the record in Line 1. According to the information, we can decrypt the ciphertext of the record and then verify the validity of the signature. Again, the function **CheckS** in Line 1 is a smart contract function which will be described in the next subsection.

Algorithm 4. VerifyStamp

```
Input (H, Record, PK)
Output Ture or False
 1: [R,Sig,C,T,sk || iv] = await CheckS(H);
 2: if $R \neq 0$ then
 3: Return False
 4: else
 5: P = Decrypt(C, sk || iv);
 6: if $P \neq Record$ or
        $H \neq SHA256(Record)$ then
 7:         return False
 8: end if
 9: Return ECDSA.verify(PK, Sig, H)
10: end if
```

To get a close view of the mentioned data we mentioned in the algorithm design, we give the instances of the data as is shown in Table 1. Note that we didn't give an instance of the items (*PK*, *SK*) due to the limit of the paper space. An original record may contain some key information such as a record identity. The length of the hash, encrypted record, signature and secret key are 256 bit, 512 bit, 560 bit and 256 bit separately. The value of the timestamp is a block timestamp which stands for the record is stamped at Tue, 21 Dec 2021 06:28:32 GMT.

Table 1. Instances of the mentioned data

Item name	Description
Record	RecordId:01111; EventId:000111; Sender: Alice; Recv:Bob; Amount:1000
Hash	0x175842f8ad5d9d0e924e62ef44cf91b404 20cfbe8e36133c7402fbcedf5c7118
Encrypted record	0x2016c29b4198fa6bf6023e569696dc8b730b4bc7c 6b599948e9d3e0e6c7dd2bafb2abd0bd304cc4063d552 6422de869c2e80be5a878f9eae02d327b5f751ee33
Signature	0x304402206e145ea4ae7ef16b6e505d0a19 f3e2c2bd2c5d1a44cee55d0630b87c5a24d7e902200 252ab5bc9ac76d64a1556b32762cbfde2f 19a20adf3c233d062e66bb4fa582d
Timestamp	0x1640058930
sk	0xe10d387a7fc28aaaab4127b0e3261413f1667101b cdc454d613251ab71dfccd6

4.3 Smart Contracts Design

Finally, we present the design of the smart contract in this sub-section. The functions of the smart contract are corresponding to those in the design of the above algorithms. Similarly, there are three primary functions, namely "**AddS**", "**UnlockS**", and "**CheckS**". Before describing these functions, we firstly have a look at the data structure in the smart contract. As is shown in Table 2, a trusted time-stamping record in a smart contract contains all the necessary information.

Table 2. The structure of a record

Data type	Description
address	owner
string	digest
string	cipher
string	signature
uint	timestamp
string	sk

Next we will give the three functions in detail as follows:

• **AddS**: As is shown in Algorithm 5, the function **AddS** is a basic operation for a user to add the information of a record into the blockchain. Note that records in Line 1 is a key-value map in the smart contract where the key is the hash value of a record and the value is a data structure in Table 2. Line 1-3 ensures that anyone cannot add a record that has already been in the blockchain. The notation *msg.sender* is an address of the transaction caller in the Ethereum network. It means there should not be two records with the same information. And we can also find that the information of the secret key *sk* doesn't add into the blockchain in this step. An example transaction of this operation is shown below. Note that we don't give all items and the whole data structure of the input as the paper space is limited.

```
{
Transaction Hash: 0xd6fa1d19f6674562d26c7eac7175f8f5
d3dd514846f9917bc55192676c47f25a,
From: 0x9eb810fd4bcc3f69cdeea333de1110257e6c91bb
To: Contract 0x9cd00b1320d0cfdbf15bb9a24199351bfdaf54df
Transaction Fee: 0.000384516003076128 Ether
Gas Price: 0.000000001000000008 Ether
Input Data: 0x842e733500000000000000000000000000
}
```

Algorithm 5. AddS

```
Input (H, C, Sig)
Output Ture or False
 1: if records[H].timestamp $\neq 0$ then
 2: Return False
 3: else
 4:    records[H].owner = msg.sender;
 5:    records[H].digest = H;
 6:    records[H].signature = Sig;
 7:    records[H].cipher = C;
 8:    records[H].timestamp = block.timestamp;
 9: end if
10: Return Ture
```

• **UnlockS**: As is shown in Algorithm 6, the function **UnlockS** is a patch operation for a user to patch the secret key *sk* of a record into the blockchain. Line 1-2 ensures that only the owner can add the *sk* to his/her record. Once the *sk* is written into the record and confirmed by the blockchain, it cannot be changed again. An example transaction of this operation is shown below. Note that we don't give all items and the whole data structure of the input as the paper space is limited.

```
    {
    Transaction Hash: 0x1e3b0b7860c933f42ef12e53afcb523d1126
    437ec09cc7385a4eaa6cae560078,
    From: 0x9eb810fd4bcc3f69cdeea333de1110257e6c91bb
    To: Contract 0x9cd00b1320d0cfdbf15bb9a24199351bfdaf54df
Transaction Fee: 0.004693 Ether
    Gas Price: 0.00000005 Ether
    Input Data: 0xe2d68ad3000000000000000000000
    }
```

Algorithm 6. UnlockS

```
Input (H, sk)
Output Ture or False
  1: if records[H].owner ≠ msg.sender or records[H].sk ≠
  0 or records[H] = 0 then
  2: Return False
  3: else
  4: records[H].sk = sk;
  5: end if
  6: Return Ture
```

• **CheckS**: As is shown in Algorithm 7, the function **CheckS** is a query operation for a user to get the information of a record from the blockchain. And then the user can verify its validity in the Algorithm **VerifyStamp**. Since it is a query operation, it doesn't need any transaction or gas fee.

Algorithm 7. CheckS

```
Input (H)
Output (R, Sig, C, T, sk)
  1: if records [digest] .timestamp = 0 then
  2: Return (100, 0, 0, 0, 0)
  3: else
  4:    R = 0;
  5:    Sig = records[H].signature;
  6:    C = records[H].cipher;
  7:    T = records[H].timestamp;
  8:   sk = records[H].sk;
  9: end if
 10: Return (R, Sig, C, T, sk)
```

5 Security Analysis

In this section, we will discuss why our scheme can resist the potential attacks in the threat models in Section 3.4.

5.1 Resistance Against Record Owner

A record owner may attempt to backdate or postdate his/her own records in the blockchain. Tampering an existed record, a record owner can resist the audit and avoid his/her duties like paying tax. However, it is difficult to backdate or post-date records in our IoE trusted time-stamping (IoETTS). The reasons are as follow:

Firstly, a rational owner has to add a timestamp by following our scheme. It means he/she should perform the **AddStamp** and **UnlockStamp** correctly to pass the verification. Then he/she may try to backdate or postdate his/her own records. Since the smart contract automatically takes the current block timestamp as the timestamp of a record, it is impossible for the owner to backdate a timestamp of the record. It is guaranteed by the chain growth property of the blockchain. However, since the owner has the original record, he/she may try to add a repeated record into the blockchain to postdate the timestamp. As we described in iV-C, records in the smart contract are stored as a key-value hash map. Thus, it is also infeasible to add a repeated record into the blockchain.

5.2 Resistance Against Adversary

Anyone could be an adversary and try to attack the scheme. For an adversary in the system, we first focus on the attack of forging a record and adding it into the blockchain earlier than the owner. Because of the transparency of the blockchain, any transaction could be seen by anyone. When an owner performs the **AddStamp** to add a timestamp of his/her record, an adversary can get the information of this record from the transaction. We assume that the adversary is a node/miner in the blockchain. We also assume that the transaction broadcast by the owner to the blockchain will be firstly received by the adversary. Then he/she can try to delay the transaction being confirmed by the blockchain by not broadcasting the transaction. And he/she will try to forge the original record by changing its ownership. He/She will broadcast a forged record to the blockchain network. And if the forged record is confirmed by the blockchain network earlier than the real one, the adversary will win in this trick.

However, it is hard for an adversary to forge a record unless the original record has been confirmed by the blockchain network. it is because the content of the record is encrypted by AES in the operation **AddStamp**. And the secret key *sk* will not be revealed by the owner until the transaction of this operation is confirmed by the blockchain network. An adversary should get the decryption of the record before the next block packing the transaction is generated by the blockchain network. In the Ethereum blockchain network, the average time for generating a block is in seconds. Thus, the adversary should break the AES in seconds. Now let's have a look at the security of the AES and then we will know that this attack is infeasible.

The first key-recovery attacks on full AES were by Andrey Bogdanov et al. in 2011 [27]. The attack is a bi-clique attack and is faster than brute force by a factor of about four. It requires 2126-2 operations to recover an AES-128 key. For AES-192 and AES-256, 2190-2 and 2254-6 operations are needed, respectively. This result has been further improved to 2126-0 for AES-128, 2189-9 for AES-192 and 2254-3 for AES-256, which are the current best results in key recovery attack against AES. This is a small gain, as a 126-bit key (instead of 128-bits) would still take billions of years to brute force on current and foreseeable hardware. And we use AES-256 in our scheme.

Another attack by an adversary is to tamper with the record after it is confirmed by the blockchain network. We can find that this attack model is just the same as the record owner attack model. And it is infeasible too. Therefore, our scheme can also resist attacks from an adversary.

5.3 Analysis for Other Design Goals

In this subsection, we will introduce how our scheme addresses the main design goals in Section 3.5.

Availability: To avoid the considerably long delay caused by uploading the transaction to the Bitcoin blockchain, we built our scheme on Ethereum, since the handling capacity of Ethereum is stronger than those of Bitcoin. This reduces the delay of uploading transactions significantly. And the fully decentralized architecture is resilient against denial of service attacks, especially those who target the time-stamping service.

Efficiency: Our time-stamping service doesn't introduce heavy computation and communication costs on both the blockchain system and users in IoE. We will give the detail of costs in Section 6. And Ethereum can ensure the blockchain system handle multiple tasks from different users simultaneously.

Integrity: In our scheme, all records are preprocessed by cryptographic functions such as SHA256, digital signature and encryption. Anyone who tries to modify a record on the blockchain will be detected.

Scalability: Our scheme is constructed based on the Ethereum blockchain, which in fact is a peer-to-peer network. A peer-to-peer network can efficiently solve the scalability problem at large sale.

Undeniability: In our scheme, the record is signed by a secret key, which is only known to its owner entity, also it is the only entity who can use it. Therefore, it cannot deny the fact that it signs the message and the undeniability is satisfied by this crytographic signature scheme.

Identification: Each entity in the IoE has an identity. More precisely, each user has its identity bound by a public key associated with it. Therefore, the system can easily recognize it.

6 Implementation and Performance

6.1 Implementation Description

Our implementation contains two parts: Web APIs and smart contracts. The Web APIs are written in JavaScript which is a programming language that adds interactivity to your website. We used the Ethereum library "ethers.js" to interact with the Ethereum blockchain in a simple way. To implement the Cryptographic functions in the scheme, we used "Crypto.js" which is a JavaScript library of crypto standards. The smart contract is written in Solidity which is an object-oriented programming language for writing smart contracts. It can be used for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

We used a PC with the OS of Ubuntu Desktop 18.04 64x to run our tests. The CPU of this computer is dual-core with intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30GHz on each. The memory is 4G. The symmetric encryption algorithm is AES- 256. The hash algorithm is SHA256. We use "secp256k1" as the asymmetric encryption elliptic to perform the digital signature.

6.2 Cost Comparison

In Chronos, basic cryptographic operations are also introduced. Now we will give the count of these operations in Chronos [12] and our scheme for comparison. Since we can choose the same cryptographic algorithms and standards in the two schemes, here we just give a comparison of the number of these operations in totally. As is shown in Table 3, our scheme IoETTS is at a lower cost.

Table 3. Count of the cryptographic operations

Scheme	Hash operations	Digital signature operations	Encryption operations
IoETTS	2	2	2
Chronos	3	3	2

6.3 Performance Analysis

In our decentralized trusted time-stamping services, we focus on three basic applications, namely creating, unlocking and verifying a timestamp. We run 100 test cases for every application to get the average results. The results on different blockchain networks are given in Table 4 to Table 5. We can find that the time for performing the cryptographic functions are costless in comparison to the operations for interacting with the blockchain. In the tables, the cryptographic functions are a digital signature, encryption and decryption. The time for each cryptographic function is about one millisecond.

Table 4. Time of creating a timestamp

Network	Total time (MS)	Digital Signature (MS)	Enc (MS)	Transaction (MS)
Local	35.87	0.71	0.02	35.14
Rinkeby	4572.06	0.66	0.02	4571.38
Ropsten	5611.40	0.68	0.02	5610.78

Table 5. Time of unlocking a timestamp

Network	Total time (MS)	Transaction (MS)
Local	23.05	23.05
Rinkeby	4817.18	4817.18
Ropsten	5149.37	5149.37

"Transaction" stands for performing a transaction on the blockchain while "Query" is just a query operation on the blockchain. Although the operations relevant to the blockchain have a more large cost than the cryptographic functions, they are still efficient and can be finished in seconds. The cost of the operations in the local Hardhat network is less than the other two test networks due to the free of network communication in the local network. Thus, we can approximately estimate the cost for the consensus mechanism in PoW and PoA.

Furthermore, we used multiple test cases to see the stability of the service in different networks. The results on different blockchain networks are given in Figure 2 to Figure 4. As we can see from the figures, the performances are quite stable in all three networks.

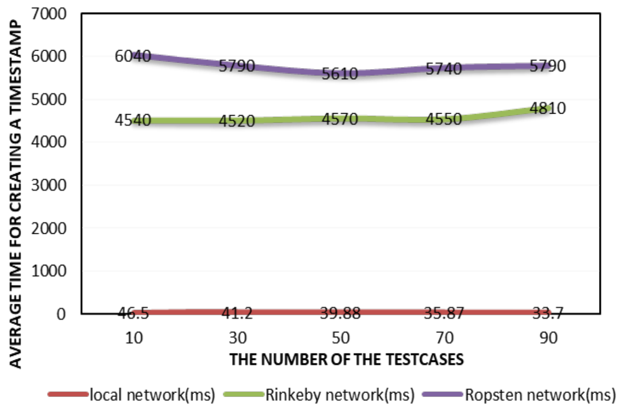


Figure 2. Time of creating a timestamp

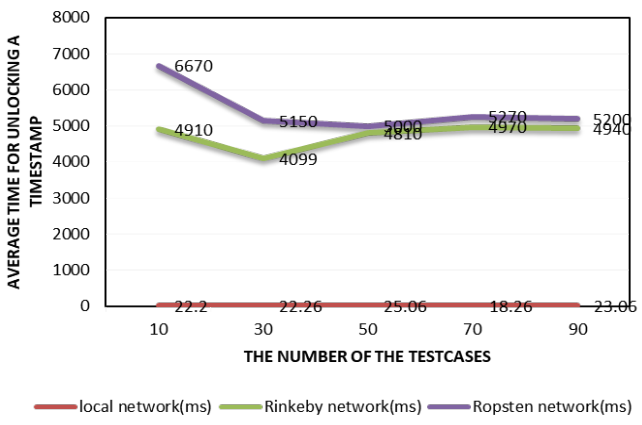


Figure 3. Time of unlocking a timestamp

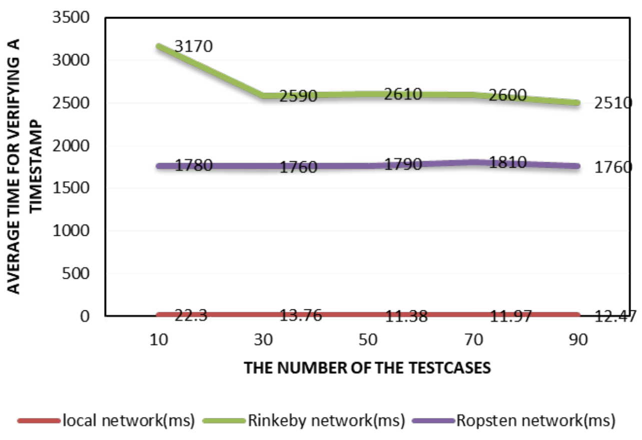


Figure 4. Time of verifying a timestamp

7 Conclusion

In this paper, we propose a fully decentralized trusted time-stamping scheme for IoE based on blockchain technology. Our scheme is a novel way to provide a trusted time-stamping service that is fit for IoE. In addition, it can reduce the security risk brought by centralization. We show in detail that how to construct the scheme and analyze the security. To avoid the long delay to confirm a transaction in Bitcoin, we construct the scheme on a more expressive blockchain system Ethereum. Finally, we give the deployment in three networks. Experiment results show the efficiency and stability of our scheme. We also give a simple comparison with another trusted time-stamping scheme. Our scheme is at a lower cost.

Recently the blockchain technology is evolving rapidly. Except for realizing trusted time-stamping services on the Ethereum blockchain, we plan to extend trusted time-stamping services to more blockchain platforms that have great new features. We will also focus on the development of IoE and improve our scheme timely to fit it.

Acknowledgement

This research was partially supported by National Key Research and Development Program of China (Grant Nos. 2019YFE0118700), Science and Technology Project of China Southern Power Grid Corporation (Grant No. 066600KK52200016).

References

[1] Internet of energy for electric mobility home page, http://www. artemis-ioe.eUioe_project.htm, Accessed: 2021-09-30.

[2] M. Guo, M. Xia, Q. Chen, A review of regional energy internet in smart city from the perspective of energy community, *Energy Reports*, Vol. 8, pp. 161-182, November, 2022.

[3] Y. R. Kafle, K. Mahmud, S. Morsalin, G. E. Town, Towards an internet of energy, *2016 IEEE International Conference on Power System Technology (POWERCON)*, Wollongong, NSW, Australia, 2016, pp. 1-6.

[4] M. Erol-Kantarci, H. T. Mouftah, Smart grid forensic science: applications, challenges, and open issues, *IEEE Communications Magazine*, Vol. 51, No. 1, pp. 68-74, January, 2013.

[5] H. Massias, X. S. Avila, J.-J. Quisquater, Timestamps: Main issues on their use and implementation, *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99)*, Stanford, CA, USA, 1999, pp. 178-183.

[6] C. Adams, P. Cain, D. Pinkas, R. Zuccherato, *Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)*, RFC3161, August, 2001.

[7] P.-Y. Ting, F.-D. Chu, Enhancing the security promise of a digital time-stamp, *22nd International Conference on Advanced Information Networking and Applications (aina 2008)*, Gino-wan, Japan, 2008, pp. 342-347.

[8] B. Gipp, N. Meuschke, A. Gernandt, *Decentralized trusted time-statmping using the crypto currency bitcoin*, iConference 2015, Newport Beach, California, USA, 2015, pp. 1-5.

[9] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, *Decentralized Business Review*, pp. 1-8, October, 2008.

[10] T. Hepp, A. Schoenhals, C. Gondek, B. Gipp, Originstamp: A blockchain-backed system for decentralized trusted timestamping, *IT Information Technology*, Vol. 60, No. 5-6, pp. 273-281, October, 2018.

[11] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, R. Wattenhofer, On scaling decentralized blockchains, *International Conference on Financial Cryptography and Data Security*, Christ Church, Barbados, 2016, pp. 106-125.

[12] Y. Zhang, C. Xu, N. Cheng, H. Li, H. Yang, X. Shen, Chronos+: An accurate blockchain-based time-stamping scheme for cloud storage, *IEEE Transactions on Services Computing*, Vol. 13, No. 2, pp. 216-229, March-April, 2020.

[13] G. Estevam, L. M. Palma, L. R. Silva, J. E. Martina, M. Vigil, Accurate and decentralized timestamping using smart contracts on the ethereum blockchain, *Information Processing & Management*, Vol. 58, No. 3, Article No. 102471, May, 2021.

[14] S. Haber, W. S. Stornetta, How to time-stamp a digital document, *Conference on the Theory and Application of Cryptography*, Santa Barbara, California, USA, 1990, pp. 437-455.

[15] A. Buldas, H. Lipmaa, B. Schoenmakers, Optimally efficient accountable time-stamping, *International Workshop on Public Key Cryptography 2000*, Melbourne, Victoria, Australia, 2000, pp. 293-305.

[16] D. Bayer, S. Haber, W. S. Stornetta, Improving the efficiency and reliability of digital time-stamping, *Sequences II: Methods in Communication, Security, and Computer Science*, Positano, ltaly, 1991, pp. 329-334.

[17] S Shamshad, Minahil, K. Mahmood, S. Kumari, C. M. Chen, A secure blockchain-based e-health records storage and sharing scheme, *Journal of Information Security and Applications*, Vol. 55, Article No. 102590, December, 2020.

[18] J. Chen, W. Gan, M. Hu, C. Chen, On the construction of a post-quantum blockchain for smart city, *Journal of Information Security and Applications*, 2021, Vol. 58, Article No. 102780, May, 2021.

[19] E. K. Wang, R. P. Sun, C. M. Chen, Z. Liang, S. Kumari, M. K. Khan, Proof of X-repute blockchain consensus protocol for IoT systems, *Computers & Security*, Vol. 95, Article No. 101871, August, 2020.

[20] C. M. Chen, X. Deng, W. Gan, J. Chen, S. K. Islam, A secure blockchain-based group key agreement protocol for IoT, *The Journal of Supercomputing*, Vol. 77, No. 8, pp. 9046-9068, August, 2021.

[21] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, *Ethereum project yellow paper*, EIP-150 REVISION, January, 2014.

[22] A. Bonnecaze, P. Liardet, A. Gabillon, K. Blibech, Secure time-stamping schemes: A distributed point of view, *Annales des telecommunications*, Vol. 61, No. 5-6, pp. 662-681, June, 2006.

[23] D. Rachmawati, J. Tarigan, A. Ginting, A comparative study of message digest 5 (MD5) and SHA256 algorithm, *2nd International Conference on Computing and Applied Informatics 2017*, Medan, Indonesia, 2017, Article No. 012116.

[24] D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ECDSA), *International journal of information security*, Vol. 1, No. 1, pp. 36-63, August, 2001.

[25] S. Heron, Advanced Encryption Standard (AES), *Network Security*, Vol. 2009, No. 12, pp. 8-12, December, 2009.

[26] J. R. Shaikh, M. Nenova, G. Iliev, Z. Valkova-Jarvis, Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained e-commerce applications, *IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, Tel-Aviv, Israel, 2017, pp. 1-4.

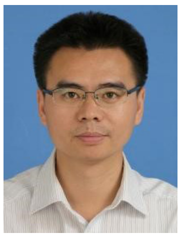
[27] A. Bogdanov, D. Khovratovich, C. Rechberger, Biclique cryptanalysis of the full AES, *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2011)*, Seoul, South Korea, 2011, pp. 344-371.

Biographies

Bin Qian received a master's degree in electrical engineering from Huazhong University of Science and Technology. He is currently the head of the measurement equipment and test laboratory, Institute of Metrology Technology, Electric Power Research Institute, CSG, Guangzhou, China. His research interests include include electrical measurement and intelligent equipment testing technology research.

Yi Luo received the B.E. degree and the Ph.D. degree in electrical engineering from Huazhong University of Science and Technology, Wuhan, China, in 2013 and 2018, respectively. He is currently a Researcher with the Electric Power Research Institute, China Southern Power Grid, Guangzhou, China. His research interests include Electric energy metering and advanced measuring technology.

Jiaxiang Ou received the Master's degree in electrical engineering from GuiZhou University, Guiyang, China, in 2010. He is currently a Researcher with the Electric Power Research Institute, Guizhou Power Grid Co.,LTD, Guiyang, China. His research interests include Electric energy metering and advanced measuring technology.



Yong Xiao received the Ph.D. degree in electrical engineering from Wuhan University, Wuhan, China, in 2016. He is currently the head of the Institute of Metrology Technology, Electric Power Research Institute, CSG, Guangzhou, China. His research interests include electrical measurement and advanced measuring technology.

Houpeng Hu received the Master's degree in electrical engineering from Guizhou University, Guiyang, China, in 2018. He is currently a Researcher with the Electric Power Research Institute, Guizhou Power Grid Co.,LTD, Guiyang, China. His research interests include online monitoring technology and Internet of Things technology.