LRCoin: Leakage-resilient Cryptocurrency Based on Bitcoin for Data Trading in IoT

Yong Yu*, Yujie Ding, Yanqi Zhao, Yannan Li*, Yi Zhao, Xiaojiang Du, Mohsen Guizani

Abstract—Currently, the number of Internet of Thing (IoT) devices making up the IoT is more than 11 billion and this number has been continuously increasing. The prevalence of these devices leads to an emerging IoT business model called Device-as-a-service (DaaS), which enables sensor devices to collect data disseminated to all interested devices. The devices sharing data with other devices could receive some financial reward such as Bitcoin. However, side-channel attacks, which aim to exploit some information leaked from the IoT devices during data trade execution, are possible since most of the IoT devices are vulnerable to be hacked or compromised. Thus, it is challenging to securely realize data trading in IoT environment due to the information leakage such as leaking the private key for signing a Bitcoin transaction in Bitcoin system. In this paper, we propose LRCoin, a kind of leakage-resilient cryptocurrency based on bitcoin in which the signature algorithm used for authenticating bitcoin transactions is leakage-resilient. LRCoin is suitable for the scenarios where information leakage is inevitable such as IoT applications. Our core contribution is proposing an efficient bilinear-based continual-leakage-resilient ECDSA signature. We prove the proposed signature algorithm is unforgeable against adaptively chosen messages attack in the generic bilinear group model under the continual leakage setting. Both the theoretical analysis and the implementation demonstrate the practicability of the proposed scheme.

Index Terms—Blockchain, Leakage Resilient Signature, Data Trading, The Generic Bilinear Group Model.

1 INTRODUCTION

The Internet of Thing (IoT) has emerged as an area of incredible potential and impact. According to Gartner Inc. [1], more than 11 billion IoT devices have been connected to the IoT network in 2018. The number of these devices is continuously increasing and is expected to be 20 billion by 2020. The application of IoT pervades everywhere from smart home, smart cities, manufacturing, commerce, education to supply chain, logistics and almost anything we can imagine [2]-[7]. The opportunities presented by IoT raise an emerging IoT business model pattern Device-as-a-service (DaaS). We can employ the sensor devices to collect data which would be vended to all interested users or devices. For example, the owner of personal weather station not only uses the IoT devices to control his household heating, but share the data to neighborhood for obtaining financial incentives.

However, most of these IoT devices are easy to be hacked or compromised by various cyber attacks such as a side channel attack. Due to this attack, the secret key in IoT devices might be leaked and then adversaries could successfully forge valid signatures to transfer bitcoins from those devices to other accounts. Actually, ten thefts of over 10,000 BTC each and more than 34 crimes of stealing accidents over 1000 BTC each since 2011 happened [8]. It was reported by Kaspersky labs that about one million infections per month of malware designed to search for secret keys and steal bitcoins were detected [8]. Dubbed Satori IoT Botnet exploits zero-day to zombify Huawei Routers and was found infecting more than 200,000 IP addresses in just 12 hours [9]. Cisco’s Talos cyber intelligence units discovered an advanced piece of IoT botnet malware, dubbed VPNFilter, that was designed with versatile capabilities to gather intelligence, interfere with internet communications, as well as conduct destructive cyber attack operations. The malware infected over 500,000 devices in at least 54 countries, most of which are small and home offices routers and internet-connected storage devices from Linksys, MikroTik, NETGEAR, and TP-Link. Some network-attached storage devices were targeted by the malware as well [10]. To sum up, the security of IoT devices was degraded due to a variety of factors, and the increasing IoT applications based on Bitcoin leads to an urgent need for more secure bitcoin transactions [11].

Blockchain was firstly introduced by Satoshi Nakamoto in Bitcoin white paper [12]. A blockchain is a hash-based data structure. Each block has a block header, a hash pointer to the previous block and a Merkle hash tree (MHT) that digests of transactions in the block. A blockchain makes use of two well-known cryptographic techniques, namely digital signatures and hash functions. A digital signature is employed to provide the integrity, non-repudiation and authentication of bitcoin transactions. A hash function is used to compute a hash value of the previous block and make the blocks as a chain. The decentralization of blockchain benefits IoT in many applications such as access control to data [13], data trading [14], and key management [15]-[18] in IoT etc.

Related Work. Noyen et al. [14] discussed how sensing-as-a-service can benefit from Bitcoin and described the process of exchanging data for cash via Bitcoin. Zhou et
al. [19] presented distributed data vending on blockchain by combining data embedding and similarity learning. This approach brings the trade-off between the effectiveness of data retrieval and leakage risk from indexing the data. Leiba et al. [20] used blockchain as a decentralized IoT software update delivery network in which participating nodes as distributors are compensated by vendors with digital currency for delivering updates to devices. Delgado-Segura et al. [21] introduced a fair protocol for data trading based on Bitcoin script language and double ECDSA. In practically, the script language operator was disabled for Bitcoin transaction. Kopp et al. [22] presented KopperCoin, a distributed file storage system with financial incentives. Later, Kopp et al. proposed privacy-preserving distributed file storage system with financial incentives [23], which takes advantages of ring signatures and one-time addresses to realize a privacy-preserving payment mechanism. However, these solutions provide no confidentiality and reliability of data in the context of side channel attacks.

Our Contributions. Blockchain is subject to side-channel attacks due to the openness of its deployment. As a consequence, information leakage especially secret key leakage is possible in various IoT applications. In this paper, we propose a new kind of cryptocurrency named LRCoin which is secure even part of the signing key of a user is exposed. LRCoin can be used in the applications where secret key leakage is inevitable such as the payment of data trading in IoT. We propose a concrete construction of an efficient bilinear pairing-based continual leakage-resilient ECDSA signature algorithm as the building block for signing transactions in LRCoin. The proposed signature algorithm is proven unforgeable against adaptively chosen messages attack in the presence of continual leakage setting. The security proof is conducted in the generic bilinear group model to bypass the impossible results that achieving continual leakage-resilience cryptographic protocols whose secret key is uniquely determined by the corresponding public key. We also implement the proposed signature algorithm on laptops and phones respectively, which demonstrates that its efficiency is comparable with that of the original ECDSA signature.

2 DATA TRADING MODEL IN IOT

Data trading [24], [25] is the exchange of bitcoin for data collected by IoT devices between data seller and data buyer. The market is to establish a platform for data seller to use blockchains as infrastructures to sell the data. The data buyer can retrieve data from blockchain and complement the payment. In this section, we introduce the data trading model in IoT and the key components. The participants involved in data trading model in IoT include Data Sellers, Trade Market (Blockchain Network, Storage Server), Data Buyers, as shown in Fig 1.

Data seller: A data seller is a user who owns the data and wants to sell the data.

Trade market: The trade market is composed of blockchain network and storage server. The blockchain network provides data sellers and data buyers a trading platform. The storage server provides data sellers and data buyers an intermediate facility to upload data and download data respectively.

Data buyer: A data buyer submits Purchase Transactions to the trade market to match the corresponding Sale Transactions. A data buyer generates a payment transaction for data she wants to buy and downloads the data from the storage server.

Sale Transaction: A data seller constructs a Sale Transaction with the topic that he has some data to sell and then broadcasts the transaction to the trade market and uploads the data to the storage server. A sale transaction consists of the topic of the data, the intended price of selling the data etc.

Purchase Transaction: A data buyer constructs a purchase transaction with the topic of the data that she wants to buy and then broadcasts the transaction to data market. A purchase transaction consists of the topic of the data, the intended price of buying the data etc.

3 PRELIMINARIES

In this section, we recall some preliminaries used in this paper, including bilinear maps, leakage-resilient models and the security model for continual leakage-resilient digital signatures.

3.1 Bilinear Maps

Let \( G = \langle g \rangle, \ G_T = \langle g_T \rangle \) be two multiplicative cyclic groups of prime order \( p \) with \( k \) bits. A map [26] \( e : G \times G \rightarrow G_T \) is called a bilinear map if the follow conditions holds.

Bilinearity. For all \( u, v \in G \) and \( a, b \in \mathbb{Z}_p, e(u^a, v^b) = e(u, v)^{ab} \).

Non-Degeneracy. \( e(g, g) \neq 1_{G_T} \), the identity element of \( G_T \).

Efficient Computation. \( e(u, v) \) can be computed in polynomial time.

3.2 Stateful Signatures

In order to achieve continual leakage resilience when a significant bits of secret key are leaked during each round of computations, it is necessary to let the secret key stateful. That is, the secret key must be refreshed after each round of signature computation. Otherwise, the secret key would eventually be completely exposed. Galindo and Vivek [27] suggested to split the secret key into two parts and reserve them on two distinct parts of a device memory. Specifically, they divide a signature computation into two steps. In each step, the memory in use is divided into two parts called the active part and the passive part. The active part of the memory is the memory being accessed by the computation while other parts of the memory are the passive part. It is
assumed that information leakage is only possible in the active part at any specified time.

There are four polynomial-time algorithms namely \( \text{KeyGen}, \text{Sign}_1, \text{Sign}_2, \text{Verify} \) in a stateful signature scheme. Different from the key generation of traditional digital signatures which generates a single secret key, the key generation algorithm in a stateful signature outputs two initial secret states \((S_0, S_1')\). The two signing algorithms \( \text{Sign}_1, \text{Sign}_2 \) are executed sequentially to generate a signature on a message \( m \). A bit more specific, the \( i \)-th round of the signature computation is performed as follows.

\[
\text{Sign}_1(S_{i-1}, m_i, r_i) \rightarrow (S_i, w_i), \\
\text{Sign}_2(S'_{i-1}, w_i, r'_i) \rightarrow (S_i, \sigma_i),
\]

where \( r_i \) and \( r'_i \) denote the random values used in \( \text{Sign}_1 \) and \( \text{Sign}_2 \) respectively, \( w_i \) denotes the state information delivered to \( \text{Sign}_2 \) by \( \text{Sign}_1 \). Then the secret state is updated from \((S_{i-1}, S'_{i-1})\) to \((S_i, S'_i)\). The signature of \( m_i \) is \( \sigma_i \).

The formal definition of a stateful signature \( \Pi = (\text{KeyGen}, \text{Sign}_1, \text{Sign}_2, \text{Verify}) \) is as follows.

- **KeyGen(\( k \))**: On input a security parameter \( k \), it outputs a key pair \((pk, (S_0, S'_0))\) where \( pk \) is public key and \((S_0, S'_0)\) are two shares of the secret key.
- **Sign_1(S_{i-1}, m_i)**: On input the first part of the \((i-1)\)-th secret key \( S_{i-1} \) and message \( m_i \), it selects a randomness \( r_i \) and updates \( S_{i-1} \) into \( S_i \) and computes the state information \( w_i \), which would be passed onto \( \text{Sign}_2 \).
- **Sign_2(S'_{i-1}, w_i)**: On input the second part of the \((i-1)\)-th secret key \( S'_{i-1} \) and state information \( w_i \), it chooses a randomness \( r'_i \) and updates \( S'_{i-1} \) into \( S'_i \) and computes the \( i \)-th signature \( \sigma_i \).
- **Verify(pk, \sigma_i, m_i)**: On input the public key \( pk \), signature \( \sigma_i \), and message \( m_i \), it outputs a bit \( b = 1 \) meaning valid, or \( b = 0 \) meaning invalid.

### 3.3 Existential Unforgeability with Leakage

A cryptographic primitive is called leakage-resilient if it is secure even the adversary additionally obtains some side-channel information, that is leakage. A variety of leakage models \([31,32,33]\) have been proposed to model side-channel attacks and formalize the security for diverse cryptographic primitives. In 2004, Micali and Reyzin \([32]\) gave a leakage model namely “only computation leaks information (OCLI)”, saying that only the secret memory which is involved in computation at that time leaks information. And they noted that the leakage amount during each computation is bounded. Otherwise, the adversary continually obtains leakage from many computations, and finally the adversary is able to obtain the full knowledge of the secret key. In 2009, Akavia et al. \([34]\) introduced a general leakage model called “bounded-memory leakage model”. In this model, an adversary is allowed to adaptively choose an efficiently computable a leakage function \( f \) and send it to a leakage oracle. The adversary obtains \( f(sk) \) from the leakage oracle where \( sk \) denotes the secret key of the target user. However, the overall output length of all the leakage functions is bounded by a parameter \( \lambda \), which is smaller than the secret key \( sk \). But this model does not cover the continuous memory leakage, which could be given rise to due to various side channel attacks. In 2010, Zvika et al. \([29]\) and Dodis et al. \([30]\) formalized a “continual-memory leakage model”. This model is similar to the OCLI model except that in this model the leakage is assumed from the entire secret memory whether or not the memory is involved in computation.

Galindo and Vivek \([27]\) presented an approach to model the leakage in a signature generation by allowing an adversary \( A \) to access to a leakage oracle \( \Gamma_{\text{secretkey}}(\cdot) \). It not only gives \( A \) signatures of messages chosen by \( A \) but also allows \( A \) to obtain leakage from the current signature computation. More precisely, let \( \lambda \) be the leakage parameter, and \( A \) is allowed to adaptively select two efficiently computed leakage functions \( f_i(\cdot) \rightarrow \{0,1\}^{\lambda} \) and \( h_i \rightarrow \{0,1\}^{\lambda} \) during every round of signature generation. Specifically, the inputs of leakage functions \( f_i \) and \( h_i \) are a part of the secret key respectively, and the outputs of leakage are denoted as \( \Lambda_i = f_i(\cdot), \Lambda'_i = h_i(\cdot) \). Galindo and Vivek noted that \( A \) can determine \( h_i \) after seeing \( \Lambda_i \). But for simplicity, they only define the leakage model in which \( f_i \) and \( h_i \) are specified along with the message \( m_i \) when they are sent to the leakage oracles.

The property of unforgeability of a stateful signature scheme \( \Pi = (\text{KeyGen}, \text{Sign}_1, \text{Sign}_2, \text{Verify}) \) with continual leakage is defined by the following game between a challenger \( C \) and an adversary \( A \).

**Setup.** The challenger \( C \) runs the key generation algorithm \( \text{KeyGen}(1^k) \) to obtain a public key pair \( pk \) and the initial secret key \( (S_0, S'_0) \). \( pk \) is given to \( A \). \( C \) sets a counter \( i = 1 \) and a set of \( \omega = \emptyset \) where \( i \) denotes the \( i \)-th round of the signature query and \( \omega \) is the set of messages which have been signed by querying the Sign-Leak oracle below.

**Sign-Leak Queries.** Given with the public parameter \( pk \), the adversaries \( A \) can query a Sign-Leak Oracle \( A^{\Gamma_{\text{secret}}} \cdot \Gamma_{\text{leak}}(\cdot)(m, f_i, h_i) \) at most \( q \) numbers of signatures of messages \( (m_1, m_2, \ldots, m_i) \in [0,1]^* \) adaptively chosen by \( A \) (\( i < q \)). When \( A \) queries the Sign-Leak Oracle, If \( f_i \neq \Lambda_i \) or \( h_i \neq \Lambda'_i \), the oracle would return \( \perp \) and then abort. Otherwise the oracle would response \( A \) with a signature \( \sigma_i \) by computing

\[
\text{Sign}_1(S_{i-1}, m_i) \rightarrow (S_i, w_i) \text{ and } \text{Sign}_2(S'_{i-1}, w_i) \rightarrow (S'_i, \sigma_i).
\]

During each such signature computation, the adversaries \( A \) could also get some knowledge about the internal secret key from the leakage functions \( f_i \) and \( h_i \) functioning by \( \Lambda_i = f_i(S_{i-1}, r_i) \) and \( \Lambda'_i = h_i(S'_{i-1}, r'_i, w_i) \). After each such query, the counter \( i \) is increased to \( i + 1 \) and the messages set is enlarged by \( \omega \cup m_i \). Eventually the Sign-Leak Oracle returns \( (\sigma_i, \Lambda_i, \Lambda'_i) \) to the adversary \( A \).

**Output.** Finally, \( A \) gives a pair \((m, \sigma)\). If there exists (1) \( \text{Verify}(pk, m, \sigma) = 1 \) and (2) \( m \notin \omega \) then the experiment returns \( b = 1 \) meaning that \( A \) has won the game. Otherwise the experiment returns \( b = 0 \) meaning that \( A \) has failed to forge a signature.

We define \( P_{\text{A}}^{\text{forge}} \) as the probability of \( A \) wins in the above game. The probability \( P_{\text{A}}^{\text{forge}} \) is taken over the coin tosses of \( A \) and \( \text{KeyGen} \).
A signature scheme \( II \) is \((\epsilon, \tau, q)\)-existentially unforgeable under adaptively chosen message attacks with continual leakage if for all \((\epsilon, \tau, q)\)-adversaries \( A \) where \( \tau \) is the most running time of \( A \) and \( Pr_{A, \text{forg}} \) is at least \( \epsilon \) and \( q \) is the most number of queries to oracle, the probability \( Pr(b = 1) \) in the experiment \( \text{Sign-Leak}_{II}(A, k, \lambda) \) is negligible (as a function of the security parameter \( k \)).

### 3.4 Generic Bilinear Group Model

The “generic algorithm” was first proposed by Shoup, in which the group elements are encoded as unique binary strings and the special properties of the encodings of the group elements are not exploited. This model was extended to the generic bilinear group [27] where a bilinear map is involved.

The specific details about the generic bilinear group model have been formalized by Galindo in [27], where representations of bilinear group elements in \( \mathbb{G} \) and \( \mathbb{G}_T \) are given by random bijective maps \( \sigma : \mathbb{Z}_p \rightarrow \Xi \) and \( \sigma_T : \mathbb{Z}_p \rightarrow \Xi_T. \) and \( \Xi \) and \( \Xi_T \) are sets of bit strings respectively. There are oracles \( O, O_T, \) and \( O_e \) that can compute the group operations in \( \mathbb{G} \) and \( \mathbb{G}_T \) and the evaluation of the bilinear map \( c. \) These oracles accept the representations in \( \Xi \) and \( \Xi_T \) as inputs and generate such representations as outputs, which are defined as follows:

\[
O(\sigma(a), \sigma(b)) = \sigma(a + b \mod p) \\
O_T(\sigma_T(a), \sigma_T(b)) = \sigma_T(ab \mod p) \\
O_e(\sigma(a), \sigma(b)) = \sigma_T(ab \mod p)
\]

The generator \( g \) of the group \( \mathbb{G} \) satisfies \( g = \sigma(1) \) and the generator \( g_T \) of \( \mathbb{G}_T \) satisfies \( g_T = \epsilon(g, g) = \sigma_T(1) \). Since the representation of \( g \) is public, thus, users can efficiently generate random elements in both \( \mathbb{G} \) and \( \mathbb{G}_T \).

We extend the generic bilinear group model [27] slightly to the asymmetric pairing setting denoted as \( G_1 \times G_2 \rightarrow G_T \) where \( G_1 \neq G_2 \). Representations of group elements in \( G_1 \), \( G_2 \) and \( G_T \) are given by random bijective maps \( \sigma_1 : \mathbb{Z}_p \rightarrow \Xi_1 \), \( \sigma_2 : \mathbb{Z}_p \rightarrow \Xi_2 \) and \( \sigma_T : \mathbb{Z}_p \rightarrow \Xi_T \), respectively. Moreover, the generator \( P_1 \) of \( G_1 \) satisfies \( P_1 = \sigma_1(1) \), and the generator \( P_2 \) of \( G_2 \) satisfies \( P_2 = \sigma_2(1) \), and the generator \( P_T \) of \( G_T \) satisfies \( P_T = \epsilon(P_2, P_1) = \sigma_T(1) \). The adversary \( A \) has access to these generic group oracles, namely \( O_1, O_2 \) and \( O_T \). Let \( \tau_1, \tau_2 \) and \( \tau_T \) be the number of queries of \( A \) to group oracles \( O_1, O_2 \) and \( O_T \). We define the query form of \( A \) and the response form of generic group oracles as follows. Let the inputs of the \( i \)th query of \( A \) to \( O_T \) be of the form \((X_i, Y_i)\), where \( X_i \) and \( Y_i \) are representations in \( \Xi_T \). And let the response of the generic group oracle \( O_T \) to the \( i \)th query be the representation \( Z_i \), such that \( Z_i = X_i \times Y_i \). For convenience, all the above mentioned representations, including inputs and outputs are denoted as \( R_1, R_2, ..., R_{\tau_T} \), meaning that \((X_1, Y_1, Z_1, X_2, Y_2, Z_2, ...) = (R_1, R_2, R_3, R_4, R_5, R_6, ...). \) The query form of \( A \) to \( O_1, O_2 \) and their response form are similar to \( O_T \), except the response of \( O_1 \) and \( O_2 \) is \( Z_i = X_i + Y_i \). Three tables \( L_T, L_1 \) and \( L_2 \) defined below are maintained to reserve these element representations obtained from \( O_1, O_2 \) and \( O_T \).

\[
L_T = \{R_1, R_2, R_3, R_4, R_5, R_6, ..., R_{\tau_T} : 1 \leq i \leq \tau_T\} \\
L_1 = \{R_1, R_2, R_3, R_4, R_5, R_6, ..., R_{\tau_1} : 1 \leq i \leq \tau_1\} \\
L_2 = \{R_1, R_2, R_3, R_4, R_5, R_6, ..., R_{\tau_2} : 1 \leq i \leq \tau_2\}
\]

Since \( A \) can also query group oracles with representations not previously appeared in the above tables, called independent representations, we introduce another three tables in order to maintain the consistencies of the representations of the group elements.

\[
Q_T = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, ..., Q_t : 1 \leq t \leq 2\tau_T\} \\
Q_1 = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, ..., Q_t : 1 \leq t \leq 2\tau_1\} \\
Q_2 = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, ..., Q_t : 1 \leq t \leq 2\tau_2\}
\]

where \( Q_1, Q_2, ..., Q_t \) are received by generic bilinear group oracles in order. It is clear that if the total query number to oracle \( O_T \) is \( \tau_T \), then \( t \leq 2\tau_2 \) because the number of independent inputs is at most twice of the number of queries. The same conclusions apply to oracles \( O_1 \) and \( O_2 \).

#### Lemma 1

An observer of the interactions between \( A \) and generic bilinear group oracles can determine, for each representations, a sequence of integers, namely \( a_1 = (a_{11}, a_{12}, ..., a_{t1}) \) with the property that \( R_t = \sum_{j=1}^{t} a_{tj} Q_j \) in group \( G_1 \) and \( G_2 \) or \( R_t = \prod_{j=1}^{t} Q_j^{a_{tj}} \) in group \( G_T \). We call \( a_t \) the combination of \( R_t \) in generic bilinear group \( G_1, G_2 \) and \( G_T \).

Brown [28] proved this lemma in a single group \( G \), and it holds naturally in \( G_1, G_2 \) and \( G_T \). Thus, lemma 1 holds.

If there exist two different combinations \( a_j \neq a_k \) satisfying \( R_j = R_k \), we call that a collision appeared in tables \( L_1, L_2 \) and \( L_T \). When a collision is found in \( L_T \), the observer can conclude

\[
u^{a_j - a_k} = 1 \mod p \tag{1}
\]

Similarly, when a collision appears in \( L_1 \) or \( L_2 \), the observer has

\[
(a_j - a_k) \cdot u = 0 \mod p \tag{2}
\]

With these equations, the observer is able to infer some knowledge of the set \( u \). Let \( r_i \in \mathbb{Z}_p \) be the unique unknown value such that \( \sigma_T(r_i) = R_i \), then the observer can obtain some information \( r_i \) where \( r_i = a_k u \) in \( G_1 \) or \( G_2 \) or \( r_i = u^{a_k} \) in \( G_T \).

#### Lemma 2 [28]

Suppose there are at most \( m \) oracle queries, the probability of a collision occurring in a generic bilinear group for \( Z_p \) is at most \( 3^{(m+1)/2}/p \).

### 4 Our Construction

#### 4.1 Basic idea

To make the original ECDSA signature scheme continual-leakage-resilient, we apply the techniques due to Galindo et al. [27]. That is, instead of managing a single secret key, the secret key is divided into two shares which are stored in different parts of the memory. The signing algorithm is divided into two steps as well. We deal with the continual leakage of the secret key by refreshing the two shares of the secret key after each signature round regularly to keep the internal secret key stateful.
4.2 Detailed construction

The details of the proposed leakage-resilient pairing-based ECDSA signature are as follows, in which \( i \) denotes the \( i \)-th round of signing.

**Setup**\((k)\): On input a security parameter \( k \), this algorithm randomly picks a pairing friendly curve \( C \) defined on the finite field \( \mathbb{Z}_p \) and outputs the corresponding pairing friendly generic groups \( G_1, G_2 \) and \( G_T \), and a bilinear map \( e: G_1 \times G_2 \rightarrow G_T \). The operations of \( G_1 \) and \( G_T \) are denoted as \(+\) and \( \times \) respectively. This algorithm chooses a base point \( P_1 \) of \( G_1 \) and a base point \( P_2 \) of \( G_2 \), and computes \( P = e(P_1, P_2) \). \( H : \{0, 1\}^* \rightarrow \mathbb{Z}_p \) denotes a secure hash function and \( f(R) = R \mod p \) denotes an almost invertible reduction function where \( R \in G_T \) defined in \([28]\). The system parameters are denoted as \( para = (G_1, G_2, G_T, P_1, P_2, P_T, e, p, C, H, f) \).

**KeyGen**\((k, para)\): On input the security parameter \( k \) and the system parameters \( para \), this algorithm randomly picks two integers \( d, l_0 \in \mathbb{Z}_p \) and computes \( S_0 = l_0 P_1, S'_0 = (d - l_0) P_1 \) and \( Q = P_T \). The public key \( pk = (P_1, P_2, P_T, Q) \)

\[ \] while the initial secret key is \( sk = (S_0, S'_0) \).

**Sign**\((i, S_{i-1}, m_i)\): On input the first part of the \((i - 1)\)-th secret key, to sign a message \( m_i \), this algorithm randomly selects integer \( l_i, t_i \in [0, p - 1] \) and computes \( S_i = S_{i-1} + l_i P_1, R_s = e(P_1, P_2) \), \( r_s = f(R_s) \), \( h_s = H(m_i) \), and \( w_i = t_i P_1 + h_s r_s S_i \).

**Verify**\((pk, m_i, \sigma_i)\): On input a message-signature pair \((m_i, \sigma_i)\) where \( \sigma_i = (r_s, s_i) \), this algorithm computes \( h_s = H(m_i) \) and \( R_s = e(s_i, P_2) \times Q \) \( \cdot (h_s r_s) \). Output a bit 0 to indicate the signature is valid if \( f(R_s) = r_s \). Otherwise, output 0 meaning the signature is invalid.

### 5 Security Proof

In this section, we prove the security of our construction against adaptive chosen message attack under continual leakage setting in the generic bilinear group model via a sequence of games. To do this, we firstly prove the security of our scheme denoted as \( \Pi \) against \((\Gamma\text{-time}; 0\text{-query})\) adversary called a passive adversary under no leakage environment through a game \( G_1 \). Then, we prove security against a \((\Gamma\text{-time}; g\text{-query})\) adversary called an active adversary under no leakage setting through a game \( G_2 \). It is clear that the attack power of the adversaries in \( G_2 \) is stronger than that of in \( G_1 \). More precisely, except the generic group oracles \( O_1, O_2 \) and \( O_T \), \( A \) can also query a signing oracle to obtain polynomial signatures of messages. Next, we prove security against an active adversary under the continual-leakage setting through a game \( G_3 \). In this game, the adversary \( A \) can not only obtain the representations of group elements and signatures of messages but also can get some leaked knowledge of the internal secret key.

#### 5.1 Proof under no leakage setting

We firstly provide the unforgeability proof of our scheme against a passive adversary, and then, we give a proof against an active adversary in this part.

### Against a passive adversary.

**Theorem 1.** If there exists an \( (\epsilon, \tau, 0) \)-adversary \( A \) that can forge a valid signature of our scheme \( \Pi \) in the generic bilinear group model, then we can construct an \( (\epsilon', \tau') \)-hash-inverter \( I_{H'} \) where

\[
\epsilon' \geq \frac{9(\tau' + 1)}{\tau^2}.
\]

**Proof.** In order to construct a hash inverter \( I_{H'} \), we use the adversary \( A \) as a sub-routine.

In the generic bilinear group model, the functions \( \sigma_T, \sigma_1 \) and \( \sigma_2 \) behave randomly. Just like hash functions are controlled by the challenger in the random oracle model, \( \sigma_T \) is controlled by \( I_{H'} \) in the generic bilinear group model. With this simulation, we describe the response of \( I_{H'} \) to group oracle queries as follows.

**Query to \( O_T \):** Upon receiving an independent input \( R_i = Q_j \) for some \( i, j \) from \( A \), the oracle \( O_T \) selects a random \( u_j \in \mathbb{Z}_p \) \( \setminus \{r_1, \ldots, r_{i-1}\} \) and sets \( \sigma_T(u_j) = Q_j \). \( O_T \) adds \( Q_j \) into the table \( Q_T \) and appends \( u_j \) to \( u \). If \( O_T \) receives a dependent input \( R_i = R_k \) for some \( k < i \), it sets \( r_i = r_k \). Before \( O_T \) outputs the response \( Z_{3i} \), it first computes \( a_{3i} = a_1 + a_{i+1} \) which are already determined from \( A \)'s two inputs. Then it computes \( r_{3i} = \prod_{j=1}^{i} u_j^{a_{3i,j}} \).

Next, it compares \( r_{3i} \) with \( \{r_1, \ldots, r_{3i-1}\} \), which are already in the table \( Q_T \). If \( r_{3i} = r_k \) for some \( k < 3i \), then \( O_T \) responds with \( R_k \). If \( r_{3i} \neq r_k \) for all \( k < 3i \) then \( O_T \) selects randomly \( R_{3i} \in \mathbb{Z}_T \setminus \{R_1, \ldots, R_{3i-1}\} \) and appends it into the table \( Q_T \).

**Query to \( O_1 \) and \( O_2 \):** \( A \)'s queries to generic group oracles \( O_1 \) and \( O_2 \) are similar to queries to \( O_T \). A difference is that \( O_1 \) and \( O_2 \) computes the pre-image as \( r_{3i} = a_{3i} \cdot u = \sum_{j=1}^{i} a_{3i,j} \cdot Q_j \).

**Description of game \( G_1 \):** We describe a reduction game from an \( (\epsilon, \tau, 0) \)-adversary \( A \) to a \( (\epsilon', \tau') \)-hash-inverter \( I_{H'} \) of the hash function \( H \). More precisely, if \( A \) can forge a signature in probability \( \epsilon \), then \( I_{H'} \) can invert \( H \) in probability \( \epsilon' \).

The input to \( I_{H'} \) is a random element \( h \in [0, p - 1] \) as a challenge, and the goal of \( I_{H'} \) is to find \( M \) such that \( H(M) = h \). To find \( M, I_{H'} \) invokes the adversary \( A \) and let it interact with a modified simulation of generic group oracle \( O_T \). When \( A \) is invoked, it makes some queries to generic group oracles. In the end, \( A \) outputs \( (M, (r_s, s)) \), where \( M \) is an arbitrary message and \( (r_s, s) \) is a signature for \( M \) valid under the public key \( Q \). Initially, we assume without loss generality that the first independent representation in table \( Q_1 \) is the base point \( P_1 \) in \( G_1 \) and \( P_1 = Q_1 = \sigma_1(1) \). And the first independent representation in table \( Q_2 \) is the base point \( P_2 \) in \( G_2 \) and \( P_2 = Q_2 = \sigma_2(1) \). In the table \( Q_T \), the first independent representation is the signer’s public key \( Q \) and \( Q = Q_1 = \sigma_T(X) \) where \( X \) denotes the discrete logarithm of the signer’s secret key.

To make sure the output message \( M \) of \( A \) is the answer of the inverter \( I_{H'} \), we introduce the following trick. We conduct a modified simulation of the generic bilinear group oracle \( O_T \). Moreover, we ensure that the modified version of the oracle \( O_T \), from \( A \)'s perspective, is indistinguishable from the standard version. Therefore, \( A \) would operate as if it were communicating with a true generic group oracle.
We modify a query from $A$ to the generic group oracle $O_T$ as follows. Recall that, all queries of $A$ to $O_T$ have an unique combination $a$. We consider that the $i^{th}$ query such that $a_{i+1}$ has the form $(-y,0)$ where $y \in Z_p$ and $a_{i+1} \neq a_i$ for $i \in [1, k]$ and $a_{2i} = a_i + a_{i+1} \neq a_k$ for $k \in [1, 3i - 1]$. We call this $i$ special. For this special $i$, we modify the way of $O_T$ generating the output $R_{3i}$. As follows. When $IH$ receives the $i^{th}$ query to $O_T$, say two inputs $(R_i, R_{i+1})$, $O_T$ sets

$$R_{3i} = g(y \cdot h^{-1} \mod p),$$

where $g$ is the probabilistic inverse of $f$, which exists since $f$ is almost invertible. Moreover, since $h$ is selected randomly and uniformly from $[0, p - 1]$, so is $y \cdot h^{-1} \mod p$. Thus, the distribution of modified simulation $R_{3i}$ is indistinguishable from the uniform distribution over $\mathbb{Z}_p$. As a result, $A$ is able to forge a signature $(r_s, s)$ for message $M$ as it normally would with the true simulation.

Upon receiving the forgery $(M, (r_s, s))$ from $A$, $IH$ computes $R_v = e(s, P_2) \times Q^{-(H(M) \cdot r_s)}$ using the method of double-and-multiply among point elements in $\mathbb{G}_T$, with additional $n$ queries to the modified simulation of the generic group oracle $O_T$. After the last query, $IH$ gives the response $R_{3(m+n)} = R_v$, which has a unique combination $a_v = a_i + a_{i+1}$.

Note that $R_v = e(s, P_2) \times Q^{-(H(M) \cdot r_s)}$, which is assumed to be the $i^{th}$ query of $A$ to $O_T$ at this moment. Next we present the response of $O_T$ to this query. Intuitively, there are two possibilities for $a_v$. If $e(s, P_2)$ is an independent representation then $a_v = (0, 1)$ and $a_v = (-H(M) \cdot r_s, 0, 0, 0, \ldots, 1)$. Else if $e(s, P_2)$ is $R_l$ for $l < \tau$, then $a_v = a_l + (-H(M) \cdot r_s)(0, a_{l+1})$. If $a_v \neq a_l$ for all $l < 3(m+n)$ then $R_v = R_{3(m+n)} \neq R_l$ for $l < 3(m+n)$. In this case, $IH$ selects a representation $R_v$ from $\mathbb{G}_T \setminus \{R_{3l}, R_{3l+1}, \ldots, R_{3(m+n)-1}\}$, since $A$'s forgery is valid, so there is $f(R_v) = r_s$. If $R_v$ is selected randomly, the probability of $f(R_v) = r_s$ is at most $3/(p-3(m+n))$. Therefore, we assume $a_{3l(m+n)} = a_l$ for $l < 3(m+n)$. In this case $e(s, P_2)$ is not an independent input. Because if it was independent, the combination $a_v = (0, H(M) \cdot r_s, 0, 0, \ldots, 1)$, which has one more integers than $a_l$ for $l < 3(m+n)$. So $a_l = a_k$ for $k < i$ and $a_v = a_l = (a_{l+1} - H(M) \cdot r_s, a_{l+2}, a_{l+3}, \ldots, a_{l+k})$. Obviously, $a_v \neq a_l$, so $R_v$ is not independent. Therefore, $R_l$ first appeared as an output, say $R_l = R_{3l}$ for $l \in [1, \ell - 2]$. As a result $a_v = a_{3l(m+n)}$. $IH$ only modifies the response of one of these queries, denoted as $j$. Moreover this $j^{th}$ query was chosen randomly by $IH$ before game. So the probability of $j = g$ is at least $1/m$, assuming there has been $m$ queries until now. Because

$$R_v = R_{3j} = g(h^{-1} \cdot H(M) \cdot r_s)$$

and

$$f(R_v) = h^{-1} \cdot H(M) \cdot r_s = r_s,$$

so $H(M) = h \mod p$ and $IH$ outputs $M$ as the solution to the given hash inverse problem, such that $H(M) = h$.

The probability $\epsilon'$ is bounded as follows. Certainly, we need $A$ to succeed, which occurs with probability $\epsilon$. Moreover, the above proof assumes no collision occurs, so the probability of collision occurs must be subtracted, which is at most $9(\frac{m+n+1}{2})/p$. What’s more, the aforementioned proof requires $j = g$, which occurs with probability at least $1/m$. So overall,

$$\epsilon' \geq \epsilon - \frac{9(\frac{m+n+1}{2})}{p} \geq \epsilon - \frac{9(\frac{m}{2})}{p} \geq \epsilon - \frac{9(\frac{\tau}{2})}{p}.$$

Because

$$\epsilon - \frac{9(\frac{\tau}{2})}{p} \geq \epsilon - \frac{9(\frac{\tau}{2})}{p} \geq \epsilon - \frac{9(\frac{\tau'}{2})}{p}$$

we get

$$\epsilon' \geq \epsilon - \frac{9(\frac{\tau'}{2})}{p}$$

Against an active adversary.

**Theorem 2.** If there exists an $(\epsilon, \tau, q)$-adversary $A$ that can forge a valid signature of the proposed scheme $\Pi$, then there exists an $(\epsilon', \tau')$-collision-finder $C_H$ where

$$\epsilon' \geq \epsilon - \frac{9(\frac{\tau'}{2})}{p}$$

**Proof.** The adversary $A$ can adaptively choose messages to query signatures to $C_H$. Let $\omega$ be the set of messages queried by $A$. For simplicity, we only describe the differences between the proof of Theorem 1 and Theorem 2.

**Description of game $G_2$:**

In this game, we aim to construct a collision finder $C_H$ using the adversary $A$. Similarly, $A$ can interact with a modified simulation of the generic group oracle $O_T$. The difference is that instead of choosing a random $j$ and modifying the response of the generic group oracle $O_T$ to the $j^{th}$ query, $C_H$ now, for each output $Z_{3i}$ of the generic group oracle $O_T$, chooses a random message $M_i$, computes $h_i = H(M_i)$ and uses $h_i$ and $g$ to generate a random output in the same way as in the proof of Theorem 1. In this proof, $A$ can interact with a signing oracle. For each signing query on a message $M_i$, $C_H$ selects a random signature $(r_{s_i}, s_i)$ and $A$ can query $O_T$ to validate this signature. We stress that $O_T$ must be simulated by $C_H$ in such a way that the queried signatures are valid, using the almost invertibility of $f$. More precisely, when $C_H$ receives a query message $M_i$, it will respond $A$ with a random signature $(r_{s_i}, s_i)$, and at the same time, $C_H$ computes $e(s_i, P_2) \times Q^{-(H(M_i) \cdot r_{s_i})}$ and stores them in the table $L_T$. Furthermore, $C_H$ sets $R_v = e(s_i, P_2) \times Q^{-(H(M_i) \cdot r_{s_i})} = g(r_{s_i})$ and puts it in the table $L_T$. Then when $A$ queries $O_T$ with $R_v$, to validate that signature, $A$ will obtain a value previously computed by $C_H$. Obviously, this signature is valid with the almost invertibility of function $f$. On the other hand, when $A$ queries $O_T$ with special inputs $X_i$ and $Y_i$, $C_H$ selects a random message $M_i$ and computes $R_v = g(H(M_i) \cdot r_{s_i})$. To achieve $f(R_v) = r_s$, there is $H(M) = H(M_i)$. So overall, under no collision environment, there are two possibilities for $A$ to forge a valid signature. One is $H(M) = H(M_i)$ and another is $H(M) = H(M)$ clearly, by the definition of a forgery, there is $M \neq M_i$. Furthermore, as the distribution of $M$ is uniform and $M$ is selected randomly, so is $M \neq M_i$.\]
Thus, if \( \mathcal{A} \) succeeds, then \( C_H \) can find two messages \((M, \tilde{M})\) or \((M, \hat{M})\) with the same hash value. The probability \( \epsilon' \) is bounded by \( \epsilon' \geq \epsilon - 9\left(\frac{\tau^2}{2}\right)/p \).

### 5.2 Proof under leakage setting

In this proof, we further strengthen the attack power of the adversary \( \mathcal{A} \). That is, in addition to group elements, signature queries of adaptively chosen messages, \( \mathcal{A} \) can also obtain leakages of secret keys used in computing those signatures.

**Theorem 3.** Let \( \mathcal{A} \) be an \((\epsilon, \tau, q)\)-adversary that can forge a valid signature of \( \Pi \), then we can construct an \((\epsilon', \tau')\)-collision-finder \( C_H \) where

\[
\epsilon' \geq \epsilon - \frac{\tau^2}{p} 2^{2\lambda}.
\] (6)

**Description of game \( G_0 \):**

In game \( G_2 \), the advantage of \( \mathcal{A} \) is bounded by its success probability conditioned on the event that no collision has occurred in the lists consisting of elements of \( G_1 \), \( G_2 \), and \( G_T \). Note that the proof under non-leakage setting in \( G_2 \) and leakage setting in \( G_3 \) would be the same conditioned on the fact that a collision has not occurred. The reason is that in the event of no collision, in order to forge a valid signature the adversary has to find a collision of the hash function. Hence, the success probability of \( \mathcal{A} \) against scheme \( \Pi \) in and not in leakage setting is same as that in the event of no collision. In the leakage setting, because \( \mathcal{A} \) has access to leakage oracles \( f_i(s_k) \) and \( h_i(s_k) \) during the \( i^{th} \) signature computation, then in adversary’s view the secret key is no longer uniformly distributed. Thus, the probability that a collision occurs in the leakage setting is increased by a factor of at most \( 2^{2\lambda} \). Hence \( \mathcal{A} \) can now cause collisions among representations in tables \( L_T, L_1 \) and \( L_2 \) with the increased probability. The output of each leakage function is at most \( \lambda \) bits. Overall, there would be at most \( 2\lambda \) bits leaked. So in the view of the adversary, the secret key would only have a min-entropy \( \log p - 2\lambda \).

According to Lemma 2, the probability of a collision occurring is increased to

\[
\frac{9\binom{m+1}{2}}{p} 2^{2\lambda}.
\] (7)

The probability \( \epsilon' \) of \( C_H \) in leakage setting is bounded as

\[
\epsilon' \geq \epsilon - \frac{9\binom{m+1}{2}}{p} 2^{2\lambda} \geq \epsilon - \frac{\tau^2}{p} 2^{2\lambda}.
\]

### 6 IMPLEMENTATIONS

In this section, we show the implementations of the proposed leakage-resilient signature algorithm.

#### 6.1 Environment

We implemented the proposed scheme on a laptop with 4.00 GB RAM, 64-bit Win 7 operating system and a phone with 4.04 GB RAM, Android 7.1.1 operating system respectively. The processors of the laptop and phone are Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz and 8 the highest 2.45GHZ respectively. The implementation was conducted with C++ projects and a powerful Miracl library, which realizes kinds of cryptographic operations such as big integers and elliptic curve group operations. And then compile the C++ projects in Visual Studio 2012.

#### 6.2 Implementation results

In the implementation, we choose the security level as AES-80 bit, and employ Cocks-Pinch curve \( y^2 = x^3 - 3x + B \), with the embedding degree 2 over a finite field \( GF(p) \) where \( p = 3 \) (mod 4).

To test the efficiency of our scheme, we executed four sub-algorithms \( \text{Setup}, \text{KeyGen}, \text{Sign} \) and \( \text{Verify} \), where \( \text{Sign} \) includes \( \text{Sign1} \) and \( \text{Sign2} \). Each sub-algorithm was run 50 times on the laptop and the phone respectively. The average running time of the sub-algorithms to sign a single message is shown in Table 1. The \( \text{Setup} \) algorithm generates the public parameters of the construction which on average consumes 23.268 ms on the laptop and 40.663 ms on the phone. For \( \text{KeyGen} \), it is obvious that the most expensive operations are 2 point multiplications in \( G_1 \) and 1 exponentiation of \( G_T \). On average, the \( \text{KeyGen} \) algorithm costs 1.156 ms on the laptop and 12.380 ms on the phone. In the signature computation, the most expensive operations are 5 point multiplications in \( G_1 \), which takes about 11.08 ms on the laptop and 39.09 ms on the phone respectively for the \( \text{Sign1} \) and \( \text{Sign2} \) algorithms together. To validate a signature, the most expensive operations in algorithm \( \text{Verify} \) are 1 bilinear map and 1 exponentiation operation. The \( \text{Verify} \) takes on average 6.083 ms on the laptop and 14.526 ms on the phone.

In the experiment, the implementation results for \( \text{Setup} \) and \( \text{KeyGen} \) algorithms are almost constant both on laptop and phone, that is on average 23.268 ms and 1.156 ms on the laptop and 40.663 ms and 12.380 on the phone. This is consistent with the empirical analysis since these two algorithms are independent of the signed messages. We also tested the time cost of the \( \text{Sign} \) and \( \text{Verify} \) algorithms by increasing the number of signed messages from 1 to 10. The implementation results on the laptop and phone are demonstrated in Fig. 23. As expected, the time cost of both the \( \text{Sign} \) algorithm and the \( \text{Verify} \) algorithm increases almost linearly with the increase of the number of signed messages. This is consistent with the theoretical analysis of the proposed scheme too since once a signed message is given, its hash value is determined and all the operations of the \( \text{Sign} \) and \( \text{Verify} \) algorithms are determined.

### 7 CONCLUSION

Information leakage especially secret key leakage is a serious threat in a number of IoT applications. In this paper, 1. Methods for constructing Pairing-Friendly Elliptic Curves, http://cacr.uwaterloo.ca/conferences/2006/ecc2006/freeman.pdf
we propose LRCoin, a kind of leakage-resilient cryptocurrency based on Bitcoin, secure even part of the signing key is exposed as a helpful supplement of Bitcoin. LRCoin can be applied to the applications where information leakage is inevitable to make the payment of data trading in IoT more reliable. The core of LRCoin is a leakage-resilient digital signature for signing transactions in the network. We propose a concrete construction of an efficient bilinear-based continual-leakage-resilient ECDSA signature algorithm as the building block of LRCoin. We prove the unforgeability of the proposed signature algorithm in the generic group model. The implementations on the laptop and the phone demonstrate the efficiency and the practicability of our proposal.

ACKNOWLEDGMENT

This work was supported by National Key R&D Program of China (2017YFB0802000), National Natural Science Foundation of China (61872229), NSFC Research Fund for International Young Scientists (61750110528), National Cryptography Development Fund during the 13th Five-year Plan Period (MMJ20170216) and Fundamental Research Funds for the Central Universities (GK201702004).

REFERENCES

[1] More connected things is used. Accessed on May. 1, 2018. [Online]. Available: http://www.gartner.com/newsroom/id/3598917.

[2] Cisco Systems, Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper, Accessed on April 20, 2018. [Online]. Available: http://tinyurl.com/zzo6766.

[3] F. Restuccia, S. D’Oro and T. Melodia, “Securing the Internet of Things: New Perspectives and Research Challenges,” J. arXiv preprint arXiv:1803.05022, 2018.

[4] K. Minbaj Ahmad and S. Khaled, “IoT security: Review, blockchain solutions, and open challenges” J. Future Generation Computer Systems., Vol. 82, pp.395-411. 2018. Accessed on Apr. 20, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17315765

[5] Meng. Yan, et al., “Securing Consumer IoT in Smart Home: Architecture, Challenges and Countermeasures,” IEEE Wireless Communications Magazine., 2018.

[6] Meng. Yan, et al., “WiVo: Enhancing the Security of Voice Control System via Wireless Signal in IoT Environment,” Proceedings of the Eighteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing., ACM, 2018.

[7] Haojin. Zhu et al., “You can jam but you cannot hide: Defending against jamming attacks for geo-location database driven spectrum sharing,” IEEE Journal on Selected Areas in Communications, vol. 34, no. 10, pp. 2723-2737, 2016.

[8] Gennaro, Rosario, S. Goldfeder, and A. Narayanan, “Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security,” International Conference on Applied Cryptography and Network Security, Springer, Cham, pp. 136-174, 2016.

[9] Satori IOT Botnet Zombify Huawei Routers. Accessed on Jul. 22, 2018. [Online]. Available: https://thehackernews.com/2017/12/satori-mirai-iot-botnet.html

[10] IOT botnet malware infects half a million routers. Accessed on Jul. 22, 2018. [Online]. Available: https://thehackernews.com/2018/05/vpnfilter-router-hacking.html

[11] X. Li et al., “A Survey on the security of blockchain systems,” Future Generation Computer Systems., 2018.

[12] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Consulted., 2008.

[13] G. Zyskind, O. Nathan and A. Pentland, “Enigma: decentralized computation platform with guaranteed privacy,” Computer Science., 2015.

[14] K. Noyen et al., “When money learns to fly: Towards sensing as a service applications using bitcoin,” J. arXiv preprint arXiv:1409.5841., 2014.

[15] X. Du, M. Guizani, Y. Xiao and H. H. Chen, “A Routing-Driven Elliptic Curve Cryptography Based Key Management Scheme for Heterogeneous Sensor Networks,” IEEE Transactions on Wireless Communications, vol. 8, no. 3, pp. 1223 - 1229, March 2009.

[16] X. Du, Y. Xiao, M. Guizani, and H. H. Chen, “An Effective Key Management Scheme for Heterogeneous Sensor Networks,” Ad Hoc Networks, Elsevier, Vol. 5, no. 1, pp 24-34, Jan. 2007.

[17] Y. Li, Y. Yu, G. Min, et al. “Fuzzy identity-based data integrity auditing for reliable cloud storage systems”: IEEE Transactions on Dependable and Secure Computing, DOI: 10.1109/TDSC.2017.2662216, 2017.

[18] Y. Yu, M. H. Au, G. Ateniese, X. Huang, W. Susilo, Y. Dai, G. Min. “Identity-Based Remote Data Integrity Checking With Perfect Data Privacy Preserving for Cloud Storage”. IEEE Trans. Information Forensics and Security, vol. 12, no. 4, pp. 767-778, 2017.

[19] J. Zhou, F. Tang and H. Zhu et al., “Distributed Data Vending on Blockchain,” J. arXiv preprint arXiv:1803.05871., 2018.

[20] O. Leiba, Y. Yitzchak and R. Bitton et al., “Incentivized Delivery Network of IoT Software Updates Based on Trustless Proof-of-Distribution,” J. arXiv preprint arXiv:1805.04282., 2018.

[21] S. Delgado-Segura, C. Pérez-Sola and G. Navarro-Arribas et al., “A fair protocol for data trading based on Bitcoin transactions,” J. Future Generation Computer Systems., 2017.

[22] H. Kopp et al., “KopperCoin-CA Distributed File Storage with Financial Incentives,” International Conference on Information Security Practice and Experience Springer International Publishing, pp. 79-93, 2016.

[23] H. Kopp et al., “Design of a Privacy-Preserving Decentralized File Storage with Financial Incentives,” IEEE European Symposium on Security and Privacy Workshops., IEEE, pp. 14-22, 2017.

[24] NZ. Aitken and D. Svetinovic, “Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams,” J. IEEE Transactions on Dependable and Secure Computing., vol. 99, pp. 1-1, 2018.
[25] F. Liang, W. Yu and D. An et al., “A Survey on Big Data Market: Pricing, Trading and Protection,” *J. IEEE Access.*, vol. 6, no. 9, pp. 15132-15154, 2018.

[26] D. Boneh and M. Franklin, “Identity-based encryption from the weil pairing,” *J. Siam Journal on Computing.*, pp. 213-229, 2003.

[27] D. Galindo and S. Vivek, “A leakage-resilient pairing-based variant of the Schnorr signature scheme,” *IMA International Conference on Cryptography and Coding.*, Springer, Berlin, Heidelberg, pp. 173-192, 2013.

[28] D R L. Brown, “The Exact Security of ECDSA,” *J. Advances in Elliptic Curve Cryptography*, 2000.

[29] Z. Brakerski et al., “Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage,” *Foundations of Computer Science IEEE.*, pp. 501-510, 2010.

[30] Y. Dodis et al., “Cryptography against continuous memory attacks” *Foundations of Computer Science Annual Symposium.*, pp. 511-520, 2010.

[31] T. Malkin, T. Teranishi and Y. Vahlis et al., “Signatures resilient to continual leakage on memory and computation” *Theory of Cryptography Conference.*, Springer, Berlin, Heidelberg, pp. 89-106, 2011.

[32] S. Micali, L. Reyzin, “Physically observable cryptography,” *Theory of Cryptography Conference.*, Springer, Berlin, Heidelberg, pp. 276-296, 2004.

[33] J. Alwen, Y. Dodis, and D. Wichs, “Leakage-resilient public-key cryptography in the bounded-retrieval model,” *Advances in Cryptology-CRYPTO 2009*, Springer, Berlin, Heidelberg, pp. 36-54, 2009.

[34] A. Akavia, S. Goldwasser and V. Vaikuntanathan, “Simultaneous hardcore bits and cryptography against memory attacks,” *Theory of Cryptography Conference.*, Springer, Berlin, Heidelberg, pp. 474-495, 2009.