Journal of Information Security and Applications 70 (2022) 103345

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Elliptic curve threshold signature scheme for blockchain

Huifang Yu *, Han Wang

School of Cyberspace Security, Xi'an University of Posts and Telecommunications, Xi'an, China

ARTICLE INFO

Keywords:
Blockchain
Threshold signature
Elliptic curve
Smart contract

ABSTRACT

Smart contract execution requires the triggering condition. Triggering condition of smart contract is the information outside the blockchain, and thus an oracle is required to provide the data services. For the security of user funds in on-chain aggregation model, each oracle must pay the gas fees to transmit off-chain data to the blockchain and trigger the smart contract, but this way easily causes the network congestion. In view of this, we propose an elliptic curve threshold signature scheme for blockchain (BC-ECTSS) to overcome the above issue. In the random oracle (RO) model, we detailedly show BC-ECTSS satisfies the existential unforgeability against the adaptive chosen-message attacks; in addition, it has the anonymity and its computation cost is relatively low.

1. Introduction

Blockchain [1] is a chained database arranged in the chronological order and achieves the security of each aspects by using the cryptographic technology. Blockchain has the characteristics of non-tampering decentralization and traceability, and it can reduce the cost of financial transactions and is expected to change the existing financial transaction mode [2]. Blockchain has very broad application prospects in internet of things [3] smart grid [4] and copyright protection [5].

Core value of the blockchain is to allow each node in network reaches a consensus on certain state. On bitcoin blockchain, the miners reach a consensus on transaction status; in addition, the miners confirm whether there is enough balance in one account to transfer to another account, and confirm whether the public key of account matches the private key. In addition to confirming the transaction status, the Ethereum blockchain can process the smart contract code logic. If triggering condition of smart contract exists on blockchain, the blockchain can easily execute the smart contract. Otherwise, the execution of smart contract relies on the input of oracle data, thus the things will become more troublesome.

In the existing decentralized oracle model, each oracle signs the response information and sends it to the smart contract of blockchain. Smart contract aggregates all data into single data point and triggers the corresponding operation, this way can effectively prevent some nodes to submit the incorrect data, but it easily causes the network congestion because each node needs to pay certain gas fee to upload the external data to blockchain, where the gas fee is the fee of transaction, smart contract execution, or data storage in Ethereum. Threshold signature can effectively solve this problem, where an oracle broadcasts the response information in signature group; the participants in signature group, who believe the response information is true, sign the information and send the signature to combiner. Combiner verifies its validity and combines t valid signatures into threshold signature. Combiner pays the gas fees and sends the threshold signature to smart contract of blockchain. Finally, smart contract verifies the validity of threshold signature. In whole process, the data is transmitted to blockchain once and only one gas fee is paid, and thus it reduces the network load and so greatly saves the cost.

1.1. Related work

Shamir [6] first proposed threshold signature using Lagrange interpolation polynomials, but it requires trusted third party (TTP) to distribute the private key. Threshold schemes [7,8] solve the dishonesty problem of participants and can ensure the public verifiability. Secure bitcoin wallets using threshold signatures [9] highlighted an application aiming at the security of personal bitcoin wallet and showed how threshold signatures can be used to achieve two-party security for wallets. Elliptic curve threshold signature schemes (ECTSS) [10] have no trusted center but at least $2t$-1 participants are required to generate legal threshold signature. (t,t) ECTSS [11] requires all participants to generate legal signature, and obtaining this signature requires $3t$-1 round. Cheng and Jia [12] devised threshold signature for blockchain electronic voting based on Asmuth-Bloom secret sharing algorithm, this scheme realizes the mutual verification function between the nodes and improves the credibility of nodes; it also supports the addition and deletion of nodes to adapt to the characteristics of large liquidity of blockchain

* Corresponding author at: Xi'an University of Posts and Telecommunications, West Chang'an Street 618, Chang'an district, Xi'an 710121, China
E-mail address: yuhuifang@xupt.edu.cn (H. Yu).

https://doi.org/10.1016/j.jisa.2022.103345

Available online 4 October 2022

nodes; the private key of node can be updated regularly to resist the mobile attacks. Certificateless threshold signcryption [13] solves the certificate management and key escrow. Threshold signature method based on multiparty computation (MPC) [14] avoids the fraudulent behavior of domain name system (DNS) operators in real scenario. Gennaro and Goldfeder [15] proposed the distributed threshold signature with no center based on zero-knowledge proof. Survey in [16] describes state-of-the-art threshold signature protocols. Doerner et al. [17] devised two-party threshold elliptic curve digital signature algorithm for multi-party key generation and signing with threshold value 2.

1.2. Contributions

For saving the cost and reducing the network congestion, we construct an elliptic curve threshold signature scheme for the blockchain (BC-ECTSS), in which the participants generate the public-private key pair and complete the signatures without TTP. BC-ECTSS cannot sign the message if the number of participants is less than the threshold value t; it also supports the key share verification and ensures the honesty of participants. In addition, it supports the functions of dynamically adding and deleting the participants, signature node is dynamically added and deleted without changing the private keys of signature group; all the participants can verify the validity of secret share without exposing the secret information. For BC-ECTSS, the difficulty of attacks is equivalent to solving the ECDL (elliptic curve discrete logarithm) problem.

1.3. Roadmap

Rest of this article is arranged as follows. Section 2 introduces the preliminaries about new scheme. Section 3 describes the algorithm definition and security model for BC-ECTSS. Section 4 devises BC-ECTSS and its correctness is analyzed in Section 5. Its security analysis is in Section 6 and its performance analysis is in Section 7. Finally, conclusion about this work is drawn in Section 8.

2. Preliminaries

2.1. Blockchain Technology

Blockchain is derived from underlying technology of Bitcoin [1], which can implement the distributed trust mechanism and conduct transactions between distrusted parties without the help of any trusted third party institution. Compared with traditional database system, it has the merits of decentralization, immutability, traceability, reliability and availability.

To realize the immutability of data, blockchain uses block-based chain structure. In blockchain, each block consists of block header and block body. Block structure is shown in Fig. 1. In block header, there are hash (PreHash), random number (Nonce), Merkel Root of previous block, where Merkel Root ensures the integrity of transaction data, and block hash can be obtained by performing SHA256 hash operations on metadata in block header. Block hashes are linked together to verify whether the block is tampered. Nonce value is unique to each block as random number. User who first finds the Nonce value will obtain the reward. Block body stores multiple transaction data received after the previous block. In order to prevent the forged or tampered transactions, each transaction information is digitally signed by transaction party, and the finished transaction will be recorded in block body.

2.2. Secret Sharing Algorithm

As shown in Fig. 2, Shamir's (t,n) secret sharing algorithm [6] divides the secret s into n secret shares, where t is the threshold value. Any t secret shares can recover the secret s, but less than t secret shares cannot recover s.

Setup. Given a set $U = \{P_1,P_2,\cdots, P_n\}$ of n nodes, key generation center (KGC) generates the secret s, threshold value $t(t \le n)$, finite field F_p with prime order p and the identity ID_i of node P_i, where $i = 1, 2, \cdots, n$ and $P_i \in U$.

Secret sharing: KGC chooses $\{a_1,a_2,\cdots, a_{t-1}\}$ and generates $(t\text{-}1)$-order polynomial:$f(x) = s + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$. KGC calculates the secret share $s_i = f(ID_i)$ and obtains $\{(ID_1,s_1),(ID_2,s_2),\cdots, (ID_n,s_n)\}$, and then sends the information pairs to relevant nodes.

Secret recovery: Select t information pairs and reconstruct $f(x)$ by Lagrange interpolation formula, where $s = f(0)$.

2.3. ECDSA

Setup: KGC generates the elliptic curve E over finite field F_p with prime order p, generator G of elliptic curve E, private key d and public key $Q = dG$.

Signature: Signer selects the random k from F_p and calculates $r = x_1 \bmod p$, where$(x_1,y_1) = kG$. Signer terminates and re-executes *Signature* if $r = 0$; otherwise, signer calculates the message digest $e = H(m)$ and partial signature $s = k^{-1}(e + dr) \bmod p$. Signer terminates and re-executes *Signature* if $s = 0$ and the signature result is (r, s) otherwise.

Verification: Verifier calculates $v = x_2 \bmod p$ and $e = H(m)$, where $(x_2, y_2) = s^{-1}(eG + rQ)$. If $v = r$ holds, the signature is valid and invalid otherwise.

2.4. ECDL problem

Definition 1. Given $(G, aG, B = aG)$, where B and G are points on elliptic curve E. It is easy to calculate B by using G and a at any time, but

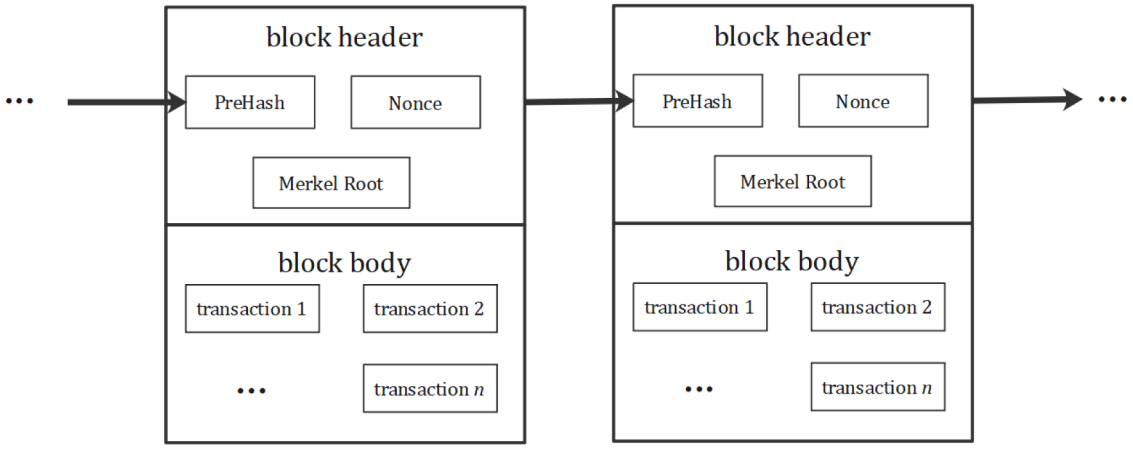

Fig. 1. Chain structure about blockchain.

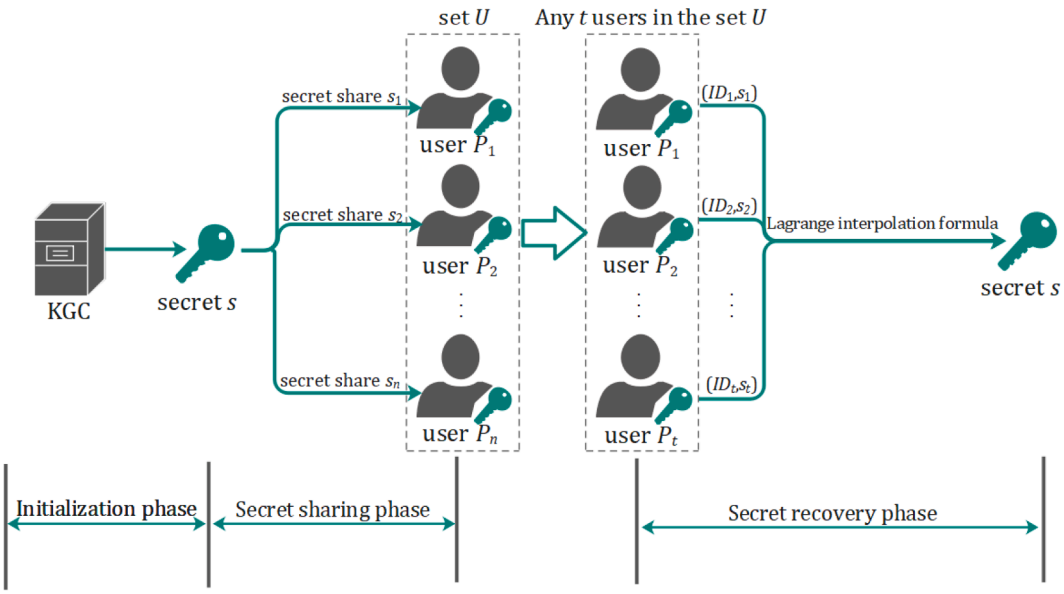

Fig. 2. Flowchart about Shamir's secret sharing algorithm.

it is difficulty to calculate a by using B and G, where $a \in Z_p$.

3. Formation definition

3.1. Symbol definition

The parameters and their meanings in BC-ECTSS as shown in Table 1.

3.2. Node definition

Blockchain nodes participating in the threshold signature involved four roles: signature group nodes, signature combiner, blockchain network nodes and verification nodes.

(1) Signature group nodes: Assume this scheme is a group composed of n nodes, each P_i ($i = 1, 2, \cdots, n$) constructs a polynomial $f_i(x)$, interacts with other nodes to calculate its secret share s_{ij}, and calculates the public-private key pair (PK_i,SK_i). Finally, P_i generates a partial signature σ_i for transaction.

(2) Signature combiner P_c: In BC-ECTSS, the signature combiner is the initiator of transaction who is the owner of message m. Verify the validity of partial signature σ_i from each node P_i, when the number more than threshold value t, transaction is combined and P_c broadcasts to all network nodes of blockchain.

Table 1
Symbols and their Implication in BC-ECTSS.

Symbol	Implication
q, p	large primes
P_i	blockchain node
ID_i	identity of blockchain node P_i
SK_i	private key of blockchain node P_i
PK_i	public key of blockchain node P_i
D	signature group private key
E	elliptic curve
G	the generator of elliptic curve E
H	hash function
T	threshold value
N	the number of signature group nodes
U	signature group set of n blockchain nodes
$f_i(x)$	t-1-order polynomial generated by node P_i
χi	Lagrange interpolation coefficient
σ_i	partial signature of blockchain node P_i
σ	threshold signature

(3) Blockchain network nodes: To construct a new block, each node collects the transactions since previous block was created and calculates the block headers. First node (miner node) found correct Nonce value will obtain the billing rights of new block, and then broadcasts this block to whole blockchain network.

(4) Verification nodes: Any verification node P_v in blockchain network can verify the validity of signature and Nonce value. If all non-miner nodes pass the verification, block is added to blockchain.

3.3. Algorithm definition

BC-ECTSS includes seven algorithms as follows. Its workflow is as shown in Fig. 3

Setup: Input a security parameter 1^γ and output the system parameter setϕ, where ϕ is used as the input of rest algorithms.

KeyGen: Input ϕ and output the public-private key pairs of all nodes.

Partial signature: Input ϕ, node identity ID_i and message m, and output the partial signature $\sigma_i = (r_i, l_i, \beta_i)$ of node P_i.

Threshold signature: Input ϕ, the identity ID_c of signature combiner and t partial signature σ_i, this algorithm verifies the validity of t partial signatures and combines them into a threshold signature $\sigma = (r, l, \beta)$.

Verification: Input ϕ, receiver identity ID_v and threshold signatures σ, this algorithm verifies the validity of σ. If the verification passes, threshold signature σ is valid and invalid otherwise.

Node addition: Input ϕ and the identity ID_a of new node $P_a(P_a \notin U)$, this algorithm adds the node P_a to the threshold signature group U and allocates its public and private keys to the node P_a.

Node deletion: Input ϕ and the identity ID_d of deleted node $P_d \in U$, this algorithm deletes the node P_d from threshold signature group and invalidates its public-private key pair.

3.4. Security model

Definition 2(unforgeability). BC-ECTSS is UF-CMA secure if no polynomial time adversary $\mathscr{A}$ wins in **UF-BC-ECTSS-CMA** with non-negligent advantage.

Security model of unforgeability relies on game **UF-BC-ECTSS-CMA** in which challenger $\mathscr{C}$ plays with adversary $\mathscr{A}$. Firstly, $\mathscr{A}$ runs *Setup*(1^γ) and outputs the relevant parameters ϕ. After that, $\mathscr{A}$ issues the adaptive queries in polynomial time:

$$\mathscr{C}_Q^{Public\ key}(ID_i) \xrightarrow{PK_i} \mathscr{A},$$

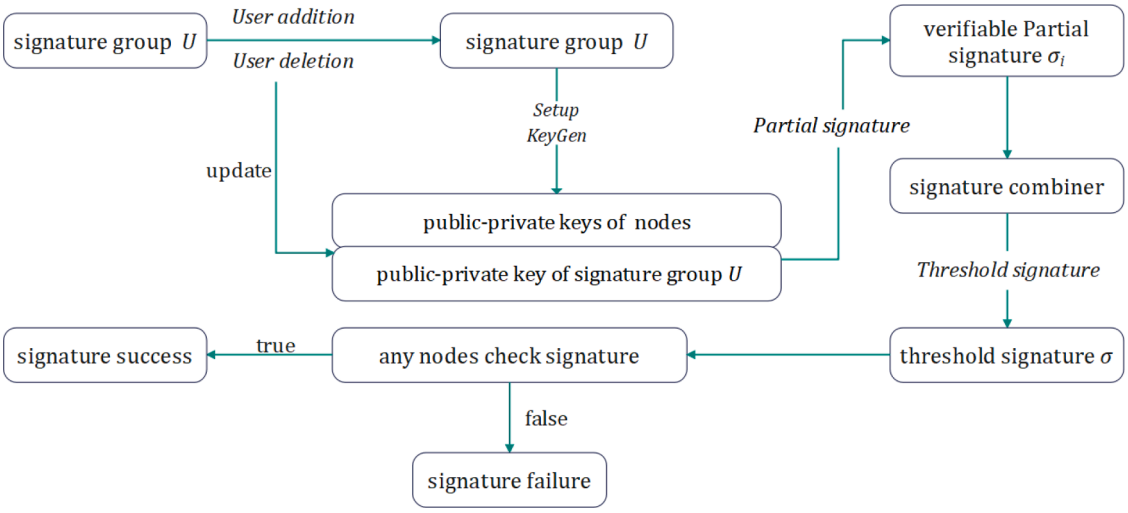

Fig. 3. Workflow about BC-ECTSS.

$$\mathscr{C}_{Q}^{Private\ \ key}(ID_i) \xrightarrow{SK_i} \mathscr{A},$$

$$\mathscr{C}_{Q}^{Threshold\ \ signature}(ID_c, ID_v, m) \xrightarrow{\sigma} \mathscr{A},$$

$$\mathscr{C}_{Q}^{Verification}(ID_c, ID_v, m) \xrightarrow{m \text{ or } \perp} \mathscr{A}.$$

Finally, $\mathscr{A}$ forges a signature $\sigma = (r, l, \beta)$. Here, if $\mathscr{A}$ can win in **UF-BC-ECTSS-CMA** and outputs a valid threshold signature σ, $\mathscr{C}$ can return the time node. Then, $\mathscr{A}$ outputs another forged threshold signature $\sigma^* = (r, l^*, \beta)$ by polynomial time queries. Based on successful forgery of different signatures, we can obtain the solution of ECDL problem by using the forking lemma [18].

Definition 3(Threshold characteristic). BC-ECTSS has threshold characteristic to satisfy the following conditions.

Signature group private key d is shared among n nodes of signature group in *KeyGen*. Only legal members greater than or equal to threshold value t can recover the signature group private key d and generate a valid threshold signature. Any subset of signature group less than t nodes cannot calculate the correct signature, that is, BC-ECTSS tolerates at most t-1 malicious signers.

Definition 4(anonymity). BC-ECTSS satisfies the anonymity with following conditions.

Given n signature nodes, message m and threshold signatures σ, even if an adversary has unlimited computing power, it cannot identify t signed nodes with better probability than random guess.

4. BC-ECTSS

4.1. Setup

KGC selects large primes p and q, p-order finite field F_p on elliptic curve E, a generator G of elliptic curve E and one-way hash functions $H: \{0,1\}^* \rightarrow Z_q^*$. $U = \{P_1, P_2, \cdots, P_n\}$, where U is a signature group set of n blockchain nodes, $t(t < n)$ is threshold value and ID_i is the identity of node $P_i \in U$. Signature group private key $d \in Z_q^*$ is shared in signature group and $Q \leftarrow dG$ is signature group public key. KGC publishes the system parameters:

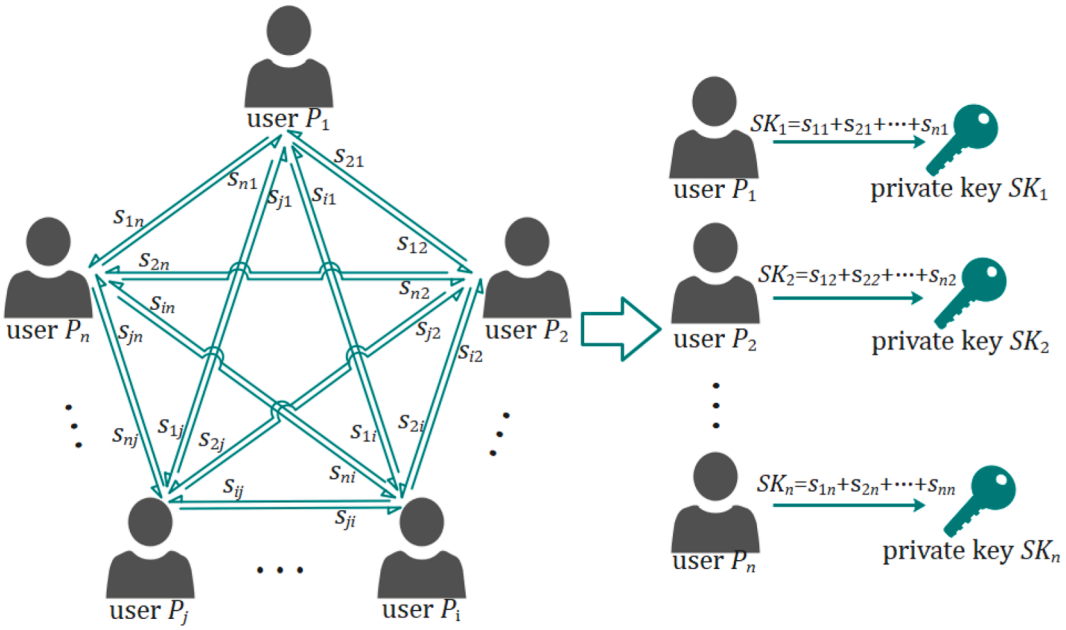

Fig. 4. Workflow about KeyGen algorithm.

$\phi = \{p, F_p, G, E, H, U, n, t\}$.

4.2. KeyGen

Workflow about *KeyGen* is as shown in Fig. 4. Concrete detail about *KeyGen* is as follows.

(1) Node P_i selects $\{a_{i0}, a_{i1}, \cdots, a_{i(t-1)}\}$ from Z_q^* and generates (t-1)-order polynomial as follows:

$$f_i(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + \cdots + a_{i(t-1)}x^{t-1}$$

(2) Node P_i calculates the secret share $s_{ij} = f_i(ID_j)$ and sends it to other nodes $P_j \in U$. Node P_i calculates $\eta_{i\mu} = a_{i\mu}G$ and broadcasts the set $\{\eta_{i0}, \eta_{i1}, \cdots, \eta_{i(t-1)}\}$ in blockchain network. where $\mu = 0, 1, \cdots, t-1$.

(3) After node P_j receives the secret share s_{ij} from other nodes and uses broadcast information $\{\eta_{i0}, \eta_{i1}, \cdots, \eta_{i(t-1)}\}$ to verify the equality $s_{ij}G = \sum_{\mu=0}^{t-1} \eta_{i\mu} ID_j^{\mu}$ if it holds, the secret share s_{ij} is valid and invalid otherwise. Node P_j calculates own public key PK_j and private key SK_j after receiving n valid secret share:

$$SK_j = \sum_{u=1}^{n} s_{uj}, \quad PK_j = SK_j \cdot G$$

(4) Signature group private key $d = \sum_{i=1}^{n} a_{i0} = F(0)$ has been determined by n nodes in group, where $F(x) = \sum_{i=1}^{n} f_i(x)$. Any node in signature group can use the broadcast information to calculate $Q = \sum_{i=1}^{n} a_{i0} G \bmod p$.

4.3. Partial signature

Signature group node P_i generates the partial signature about message m as follows:

(1) Node $P_i \in U$ randomly selects k_i from finite field Z_q^* and calculates the message digest $e = H(m)$ and as follows:
(2) P_i calculates $r_i = x_i \bmod p$, where $(x_i, y_i) = k_i G$. If $r_i = 0$, return (1).
(3) P_i randomly selects (α_i, β_i) from Z_q^* such that $k_i = \alpha_i r_i + \beta_i m$ and calculates $l_i = \alpha_i r_i + e\chi_i \cdot SK_i$, where the Lagrange interpolation coefficient $\chi_i = \prod_{j=1, j\neq i}^{t} \frac{-ID_j}{ID_i - ID_j}$.
(4) Partial signature is $\sigma_i = (r_i, l_i, \beta_i)$, node P_i broadcasts the partial signature σ_i in blockchain network and waits for further verification by the signature combiner.

4.4. Threshold signature

Workflow about partial signature and threshold signature is as shown in Fig. 5. In this algorithm, signature combiner P_c receives the t valid partial signatures σ_i and combines t partial signatures into a threshold signature σ.

(1) P_c calculates $\gamma_i = (l_i + \beta_i m) \bmod p$, $v_i = x_i' \bmod p$, $(x_i', y_i') = \gamma_i G - e\chi_i PK_i$, where $e = H(m)$.
(2) P_c verifies whether $v_i = r_i$. Partial signature σ_i from node P_i is valid if it holds and invalid otherwise.
(3) P_c receives t valid part signatures σ_i and continues to calculate:

$$r = \sum_{i=1}^{t} r_i, \beta = \sum_{i=1}^{t} \beta_i, l = \sum_{i=1}^{t} l_i.$$

(4) P_c broadcasts the threshold signature $\sigma = (r, l, \beta)$ in blockchain network, and any verifier can check the validity of threshold signature.

For constructing one new block, each node in blockchain network collects the transactions from previous block and calculates the block headers. First node (miner node) to find the correct Nonce value will obtain the billing rights of new block, and then broadcasts this block to whole blockchain network.

4.5. Verification

In this algorithm, the signature verifier P_v verifies the validity of threshold signature σ from signature combiner P_c. Concretely, P_v calculates $e = H(m)$, $\gamma = (l + \beta m) \bmod p$, $(x', y') = \gamma G - eQ$, $v = x' \bmod p$.

After that, P_v checks whether $r = v$. If it holds, the threshold signature σ is valid and invalid otherwise.

4.6. Node Addition

In this algorithm, new node $P_a \notin U$ is added to signature node group. During execution of this algorithm, it is necessary to meet the following conditions: the private key of existing node will not be changed or leaked; the addition of new node cannot change the signature group private key d; new node can obtain own public-private key pair (PK_a,

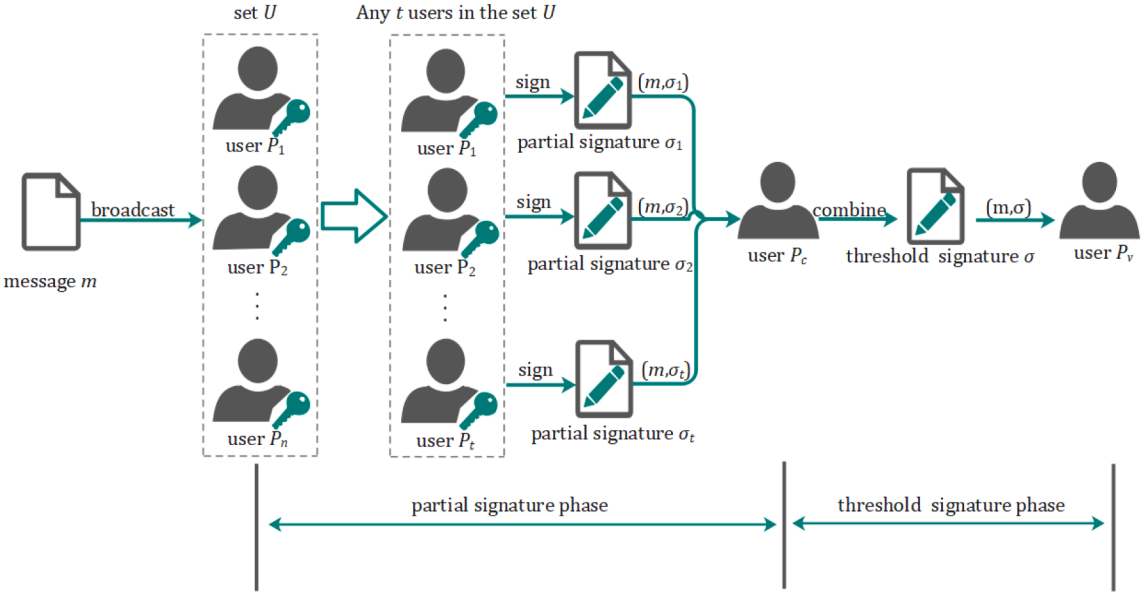

Fig. 5. Workflow about partial signature and threshold signature.

SK_a) and cooperate with existing nodes to generate valid threshold signature. Steps of adding node are as follows:

Node P_i selects $\{b_{i0},b_{i1},\cdots, b_{i(t-1)}\}$ from finite field Z_q^* and generates (t-1)-order polynomial as follows and then makes it satisfy $g_i(ID_a) = 0$.

$$g_i(x) = b_{i0} + b_{i1}x + b_{i2}x^2 + \cdots + b_{i(t-1)}x^{t-1}.$$

(2) Node P_i calculates the temporary secret share $s_{ij}^{'} = g_i(ID_j)$ and sends it to other nodes P_j, and then node P_i calculates $\theta_{i\mu} = b_{i\mu}G(\mu = 0, 1, \cdots, t-1)$ and broadcasts $\{\theta_{i0},\theta_{i1},\cdots, \theta_{i(t-1)}\}$ in blockchain network.

(3) After receiving the temporary secret share $s_{ij}^{'}$ from other nodes, the node P_j verifies the equality by using broadcast information $\{\theta_{i0},\theta_{i1},\cdots, \theta_{i(t-1)}\}$ as follows:

$$s^{'}_{ij}G = \sum_{\mu=0}^{t-1} \theta_{i\mu}\cdot ID_j^{\mu},$$

if it holds, this means that the temporary secret share $s_{ij}^{'}$ is valid and invalid otherwise.

(4) After receiving the n temporary secret share from other nodes, the node P_j calculates temporary private key $SK^{'}_j = SK_j + \sum_{u=1}^{n} s^{'}_{uj}$ and then sends it to new node P_a by using secure channel.

(5) After receiving t valid temporary private key, new node P_a calculates own private key SK_a and public key PK_a as follows:

$$SK_a = \sum_{i=1}^{t} \prod_{j=1,j\neq i}^{t} \frac{ID_a - ID_j}{ID_i - ID_j}\cdot SK_i^{'}\ ,\ \ PK_a = SK_a\cdot G.$$

4.7. Node Deletion

In this algorithm, the node $P_d \in U$ is deleted from signature group. During the execution of this algorithm, it is necessary to satisfy the following conditions: reconstruct the public and private keys of all nodes and invalidate the public-private key pair (PK_d,SK_d) of deleted node; the deletion of the node cannot change the signature group private key d. Steps of deleting node are as follows:

(1) Node P_i selects $\{c_{i1},\cdots, c_{i(t-1)}\}$ from finite field Z_q^* to construct (t-1)-order polynomial

$$h_i(x) = c_{i1}x + c_{i2}x^2 + \cdots + c_{i(t-1)}x^{t-1}.$$

(2) Node P_i calculates temporary secret share $s''_{ij} = h_i(ID_j)$ and sends it to other node P_j, and then P_i calculates $\varphi_{i\mu} = c_{i\mu}G(\mu = 0, 1, \cdots, t-1)$, and broadcasts $\{\varphi_{i0},\varphi_{i1},\cdots, \varphi_{i(t\ -\ 1)}\}$ in blockchain.

(3) After receiving the temporary secret share s''_{ij} sent from other node, node P_j verifies the equality $s''_{ij}G = \sum_{\mu=0}^{t-1}\varphi_{i\mu}ID_j^{\mu}$ by using broadcast information $\{\varphi_{i0},\varphi_{i1},\cdots, \varphi_{i(t-1)}\}$, if it holds, this means that the temporary secret share s''_{ij} is valid and invalid otherwise.

(4) After receiving n temporary secret share, the node P_j updates his new public key PK''_j and private key SK''_j as follows:

$$SK^{''}_j = SK_j + \sum_{u=1,u\neq d}^{n} s^{''}_{uj},\ \ PK^{''}_j = SK^{''}_j G$$

5. Correctness analysis

5.1. Secret share verification

In the key generation algorithm, the node verifies the validity of secret shares s_{ij} by checking the following equality, if it holds, secret share s_{ij} is valid and invalid otherwise. Concrete derivation process of secret share verification is as follows:

$$\begin{aligned} s_{ij}G &= f_i(ID_j)G = a_{i0}G + a_{i1}G\cdot ID_j^1 + a_{i2}G\cdot ID_j^2 + \cdots + a_{i(t-1)}G\cdot ID_j^{t-1} \\ &= \sum_{u=0}^{t-1} \eta_{iu}\cdot ID_j^{u} \end{aligned}$$

5.2. Partial signature verification

In threshold signature, combiner verifies the validity of partial signature σ_i, concrete derivation process of partial signature verification is as follows:

$$\begin{aligned} (x_i, y_i) &= k_iG \\ &= (\alpha_i r_i + \beta_i m)G \\ &= (\alpha_i r_i + \beta_i m)G + Ge\cdot SK_i\cdot\chi_i - Ge\cdot SK_i\cdot\chi_i \\ &= G(\alpha_i r + e\cdot SK_i\cdot\chi_i) + \beta_i mG - Ge\cdot SK_i\cdot\chi_i \\ &= l_iG + \beta_i mG - Ge\cdot SK_i\cdot\chi_i \\ &= \gamma_iG - Ge\cdot SK_i\cdot\chi_i \\ &= \gamma_iG - e\cdot PK_i\cdot\chi_i \\ &= \left(x_i^{'}, y_i^{'}\right) \end{aligned}$$

where$r_i = x_i \mathrm{mod} p$, $v_i = x_i^{'}\mathrm{mod}p$. If $r_i = v_i$ holds, this means that the partial signature σ_i is valid and invalid otherwise.

5.3. Threshold signature verification

In verification algorithm, the signature verifier verifies the validity of threshold signature σ, the concrete derivation process of threshold signature verification is as follows:

$$\begin{aligned} (x, y) &= \sum_{i=1}^{t}(x_i, y_i) \\ &= \sum_{i=1}^{t} k_iG \\ &= \sum_{i=1}^{t}(a_i r_i + \beta_i m)G \\ &= G\sum_{i=1}^{t}(a_i r_i + \beta_i m) + eG\sum_{i=1}^{t} SK_i\cdot\chi_i - eG\sum_{i=1}^{t} SK_i\cdot\chi_i \\ &= G\sum_{i=1}^{t}(a_i r_i + e\cdot SK_i\cdot\chi_i) + G\sum_{i=1}^{t}\beta_i m - eG\sum_{i=1}^{t} SK_i\cdot\chi_i \\ &= G\sum_{i=1}^{t} l_i + \sum_{i=1}^{t}\beta_i m - eG\sum_{i=1}^{t} SK_i\cdot\chi_i \\ &= G\sum_{i=1}^{t}(l_i + \beta_i m) - eG\sum_{i=1}^{t} SK_i\cdot\chi_i \\ &= (l + \beta m)G - edG \\ &= \gamma G - eQ \\ &= (x', y') \end{aligned}$$

where $r = x \mathrm{mod} p$ and $v = x'\mathrm{mod}p$. If $r = v$ holds, threshold signature σ is valid and invalid otherwise.

6. Security analysis

6.1. Unforgeability

Theorem 1. In RO model, if the underlying elliptic curve threshold signature scheme can be proved to meet the unforgeability, BC-ECTSS is also meet the unforgeability. Assume UF-CMA security of BC-ECTSS can be broken by adversary $\mathscr{A}$ with advantage ε in RO model, then an challenger $\mathscr{C}$ can solve the ECDL problem with advantage $\varepsilon' \geq \varepsilon/q_H q_k$, where$q_H$ is the query time to H oracle and q_k is the query time to public key oracle.

Proof. $\mathscr{C}$ receives a random instance (G, aG) of ECDL problem and wants to output the solution a. In **UF-BC-ECTSS-CMA**, $\mathscr{A}$ plays the role of subroutine of $\mathscr{C}$. Initially empty lists (L_H,L_k) record the query-answer values of H oracle and public key oracle. $\mathscr{C}$ chooses the challenge identity ID_τ, where $\tau \in \{1, 2, \cdots, q_H\}$ is an integer and q_H is the query time

to H random oracle.

Concrete interaction process is as follows: First of all, $\mathscr{C}$ generates $\phi \leftarrow Setup(1^{\gamma})$ and then outputs ϕ. Then, $\mathscr{A}$ issues a series of polynomial time queries in an adaptive way.

***H* queries**: At any time, $\mathscr{A}$ may issues a query relevant to H random oracle, $\mathscr{C}$ returns ξ_i to $\mathscr{A}$ if relevant record exists in L_H; otherwise, $\mathscr{C}$ stores $(m, \xi_i \leftarrow Z_q^*)$ in L_H and outputs ξ_i to $\mathscr{A}$.

Request public key: At any time, $\mathscr{A}$ may request a public key of ID_i. If relevant public key PK_i exists in L_k $\mathscr{C}$ outputs PK_i to $\mathscr{A}$; otherwise, $\mathscr{C}$ thinks about two cases as follows:

(1)If $ID_i \neq ID_\tau$, $\mathscr{C}$ returns $PK_i \leftarrow SK_i \cdot G$ to $\mathscr{A}$ and stores (ID_i, SK_i, PK_i) in L_k, where $SK_i \leftarrow Z_q^*$

(2)If $ID_i = ID_\tau$, $\mathscr{C}$ returns $PK_i \leftarrow aG$ and stores $(ID_i, -, PK_i)$ in L_k

Private key queries: At any time, $\mathscr{A}$ may request a private key of ID_i, $\mathscr{C}$ fails and outputs $\perp$ if $ID_i = ID_\tau$; otherwise, $\mathscr{C}$ obtains SK_i from L_k and outputs SK_i to $\mathscr{A}$.

Threshold signature queries: At any time, $\mathscr{A}$ may request a threshold signature to message m. $\mathscr{C}$ returns a signature σ via threshold signature algorithm if it is not τ-th query; otherwise, $\mathscr{C}$ selects $k_i \in Z_q^*$ to calculate $(x_i, y_i) = k_i G$ *and* $r_i = x_i \bmod p$. $\mathscr{C}$ selects (α_i, β_i) from Z_q^* to satisfy $k_i = \alpha_i r_i + \beta_i m$. $\mathscr{C}$ continues to calculate $l_i = \alpha_i r_i + e\chi_i SK_i$ and $\sigma = (r, l, \beta)$ as follows:

$$r = \sum_{i=1}^{t} r_i, \beta = \sum_{i=1}^{t} \beta_i, l = \sum_{i=1}^{t} l_i,$$

where (m, ξ_i) is stored in L_H. $\mathscr{C}$ outputs threshold signature $\sigma = (r, l, \beta)$ to $\mathscr{A}$.

Verification queries: At any time, $\mathscr{A}$ can request a verification query to signature σ, $\mathscr{C}$ returns a signature σ to $\mathscr{A}$ by a call to verification algorithm if it is not τ-th query; otherwise, $\mathscr{C}$ calculates $(x', y') = \gamma G - \xi_i Q$, $v = x' \bmod p$. $\mathscr{C}$ outputs m to $\mathscr{A}$ if $r = v$ and $\perp$ otherwise.

At the end of adaptive queries, $\mathscr{A}$ outputs a forgery signature $\sigma = (r, l, \beta)$. In adaptive queries, the forgery signature cannot be returned by a threshold signature oracle. $\mathscr{C}$ fails if $ID_v^* \neq ID_\tau$; otherwise, $\mathscr{A}$ outputs another signatures $\sigma^* = (r, l^*, \beta)$ in polynomial time. If the used identities of t nodes generating the threshold signature are $ID_1, ID_2, \cdots, ID_{t-1}$, $ID_t = ID_c^* = ID_\tau$, we can obtain the equations as follows based on forking lemma [18]:

$$\begin{cases} l = (k - \beta m) + eQ \\ l^* = (k - \beta m) + e^* Q \\ Q = \sum_{i=1}^{t} SK_i \chi_i G \end{cases}$$

Then, we make use of above equations to obtain the solution a of ECDL problem instance as follows:

$$a = SK_\tau = \frac{l - l^*}{(e - e^*)\chi_\tau} - \frac{\sum_{i=1, i\neq\tau}^{t} SK_i \chi_i}{\chi_\tau}.$$

Probability analysis. Assume δ is the probability of $ID_i = ID_\tau$. Based on above query process, the probability that $\mathscr{C}$ does not fail in adaptive queries is $1/q_H q_k$ [13]. Hence, the probability of $\mathscr{C}$ in solving the ECDL problem in polynomial time is at least $\varepsilon' \geq \varepsilon / q_H q_k$ which is negligible, that is, BC-ECTSS is unforgeable (UF-CMA secure).

6.2. Threshold characteristic

Theorem 2. Our BC-ECTSS meets the threshold characteristic.

Proof. BC-ECTSS distributes the signature group private key d to n nodes, only legal nodes more than or equal to threshold value t can sign validly, and the amount of nodes less than threshold value t cannot recover the signature group private key d or calculate the correct signature result. Even if attacker obtains the private keys of t-1 nodes, it can only construct t-1 equations with t unknowns $a_i (i = 0, 1, \cdots, t-1)$, as follows:

$$\begin{cases} a_0 + a_1(ID_1) + a_2(ID_1)^2 + \cdots + a_{t-1}(ID_1)^{t-1} = SK_1 \\ a_0 + a_1(ID_2) + a_2(ID_2)^2 + \cdots + a_{t-1}(ID_2)^{t-1} = SK_2 \\ \vdots \\ a_0 + a_1(ID_{t-1}) + a_2(ID_{t-1})^2 + \cdots + a_{t-1}(ID_{t-1})^{t-1} = SK_{t-1} \end{cases}$$

Because the number of equations is less than unknown numbers, and then the above equations have no solution, and the attacker cannot obtain the specific form of the equations as follows:

$$F(x) = \sum_{i=1}^{n} f_i(x) = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1}.$$

Thus, it cannot obtain the signature group private key $d = a_0 = F(0)$ and also cannot calculate the correct signature result. If the attacker wants to use Q to recover the signature group private key d to obtain a legal signature result, attacker needs to obtain d from $Q = dG$, where obtaining d is equivalent to solving the ECDL problem. Obviously, the threshold signature can be effectively generated only when at least t nodes are satisfied. Hence, BC-ECTSS satisfies the threshold characteristics.

6.3. Anonymity

Theorem 3. Our BC-ECTSS meets the anonymity.

Proof. Anonymity means that for any threshold signature, the signature verifier cannot figure out which nodes generated the threshold signature. In BC-ECTSS, the signature verifier P_v wants to know which nodes signed the message m. In signature process, k_i is a random selected from finite field Z_q^* and $(x_i, y_i) = k_i G$, so $r = \sum_{i=1}^{t} r_i = \sum_{i=1}^{t} x_i \bmod p$ satisfies the uniform distribution. In addition, the random β_i and $l_i = \alpha_i r_i + e\chi_i SK_i$ also satisfy the uniform distribution, where e is a hash digest, χ_i is a constant, α_i is a random selected from finite field Z_q^*. In summary, each partial signature of threshold signature σ satisfies the uniform distribution. Thus, the signature verifier P_v cannot figure out which nodes generated the signature. Hence, our BC-ECTSS satisfies the anonymity.

7. Performance analysis

Performance comparison between BC-ECTSS and similar schemes [19,20,21] is described in terms of computational complexity. Experiment environment is as follows: the processor is Intel(R) Core(TM) i5-6200U CPU @2.30Hz; the operating system is 64-bit windows 10. Based on this experiment environment, VC programming language calls PBC library under VMware platform to calculate the running time of each operation.

Table 2 shows the various operations types and the corresponding running time. Table 4 shows the computational efficiency and feature comparison of several schemes.

In Table 3, communication overhead is shown by using the private key size, public key size and signature length, where $|F_p|$ is the length of an element in finite field F_p, $\left|Z_q^*\right|$ is the length of an element in Z_q^*. Referring to standard compression technology in [22], when we use the 80-bits security level, finite field F_p with order q defined over elliptic curve $y^2 = x^3 + ax + b \bmod p$, where p is 512 bits and q is 160 bits. Size of an element in F_p can be reduced to 65 bits and an element in Z_q^* has the size of 20 bits. In turn, the signature length of scheme [19], scheme [20],

Table 2
The time of each cryptography operation.

Symbols	Meaning	Operation time (ms)
T_{EX}	Perform an exponential operation	5.086
T_M	Perform a scalar multiplication	0.002
T_E	Perform a scalar multiplication on elliptic curve	7.984
T_I	Perform an inverse operation	2.011
T_P	Perform a bilinear pairing operation	20.15

Table 3
Comparison of the communication overhead.

Schemes	Private key size	Public key size	Signature length
Scheme [19]	$\|Z_q^*\|$	$\|F_p\|$	$2\|Z_q^*\|+3\|F_p\|$
Scheme [20]	$\|Z_q^*\|$	$\|F_p\|$	$\|Z_q^*\|+2\|F_p\|$
Scheme [21]	$\|Z_q^*\|$	$\|Z_q^*\|+\|F_p\|$	$\|Z_q^*\|+2\|F_p\|$
BC-ECTSS	$\|Z_q^*\|$	$\|F_p\|$	$\|Z_q^*\|+2\|F_p\|$

Table 4
Comparison of computational performance and feature.

Schemes	Signature time	Verification time	Support for adding and deleting nodes	Suitable for blockchain
Scheme [19]	$3tT_M+3tT_E+tT_{EX}$	$T_M+2T_E+T_I$	×	√
Scheme [20]	$(3t+1)T_M+(4t)$	T_M+2T_E	×	×
Scheme [21]	$(t+1)T_P+(5t+7)T_E$	T_P	×	×
BC-ECTSS	$6tT_M+3tT_E$	$2T_M+2T_E$	√	√

scheme [21] and BC-ECTSS are as follows:

$$\left(2\left|Z_q^*\right|+3\left|F_p\right|\right)\text{bits}=(2\times 20+3\times 65)\text{bytes}=235\text{bytes},$$

$$\left(\left|Z_q^*\right|+2\left|F_p\right|\right)\text{bits}=(20+2\times 65)\text{bytes}=150\text{bytes},$$

$$\left(\left|Z_q^*\right|+2\left|F_p\right|\right)\text{bits}=(20+2\times 65)\text{bytes}=150\text{bytes},$$

$$\left(2\left|Z_q^*\right|+\left|F_p\right|\right)\text{bits}=(2\times 20+65)\text{bytes}=105\text{bytes}.$$

As shown in Fig. 6, we use the same method to summarize the signature length of several schemes at 112-bits security level and 128-bits security level respectively.

For operating efficiency comparison of several schemes more intuitively, we make the performance comparison chart for threshold signature time and verification time based on simulation experiment data. We define the concrete threshold value t in performance comparison. Fig. 7 is the comparison of threshold signature. As shown in Fig. 7, with the increasing of threshold value t, the time-consuming of threshold signatures of several schemes increases linearly, but the growth trend of BC-ECTSS is very slow. Fig. 8 is the comparison of verification algorithm. As shown in Fig. 8, the verification time-consuming has nothing to do with threshold value t, and the time-consuming of BC-ECTSS is less than other schemes. When the threshold values is 10,20,30, 40,50,60, BC-ECTSS has high computation efficiency.

8. Summary

Blockchain can provide the important technical support for internet of things, banking, finance, medical and other industries. For better performance, efficient BC-ECTSS is devised in this article. Using BC-ECTSS, off-chain oracles communicate with each other and reach the consensus to determine the authenticity of off-chain data sources, off-chain oracle aggregates the data and transmits off-chain data to the blockchain. In whole process, data is transmitted to the blockchain once, so only one gas fee is paid, thus, it reduces the network load and greatly saves the cost.

BC-ECTSS cannot resist the quantum attacks, so in later work, we devised blockchain-based threshold scheme with anti-quantum security and submitted to other journal. For improvement of scheme security, administrator should choose large threshold value to prevent the network attacks, but this method may bring the risk of denial of service; using the firewall can resist this kind of attacks.

Author Statement

Huifang Yu: Writing of abstract and introduction, Algorithm definition of BC-ECTSS, Construction of security models of BC-ECTSS, Instance design of BC-ECTSS, Security proof of BC-ECTSS.

Han Wang: Algorithm definition of BC-ECTSS, Instance design of BC-ECTSS, Security proof of BC-ECTSS, Matlab experiments in performance analysis section, Probability analysis

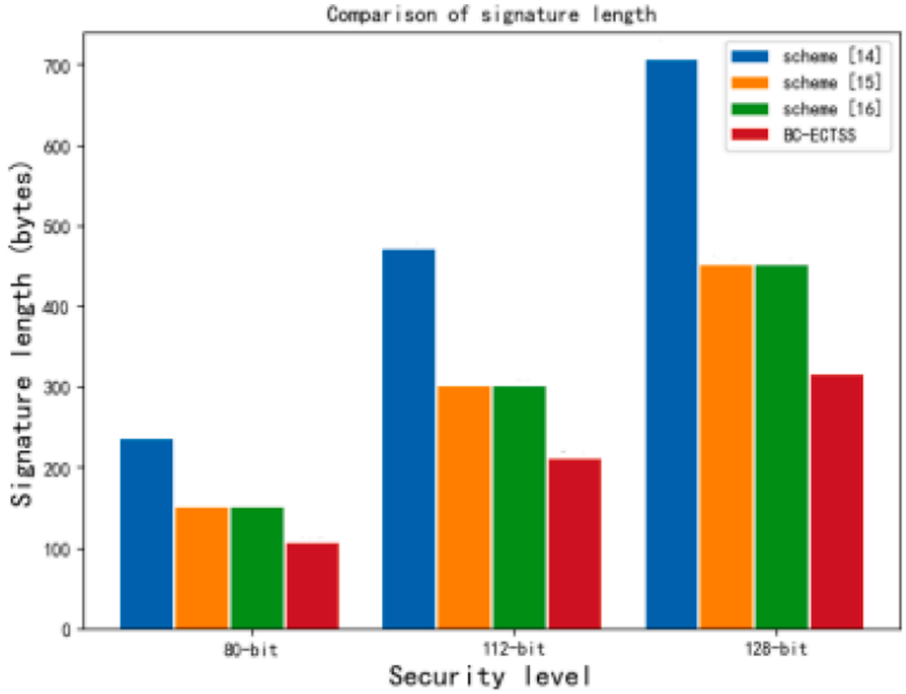

Fig. 6. Comparison of signature length.

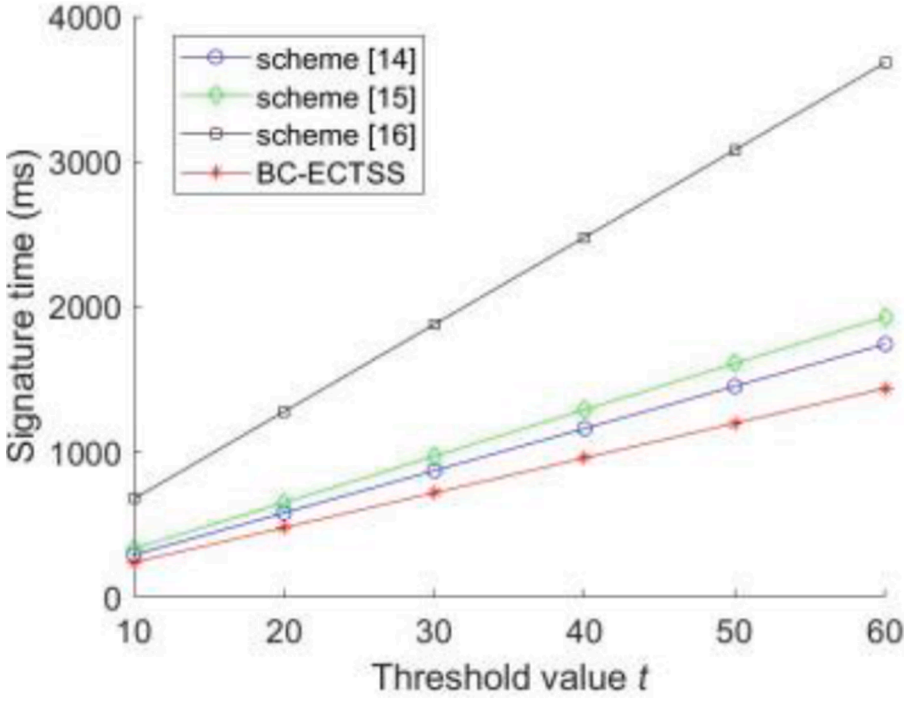

Fig. 7. Signature time comparison.

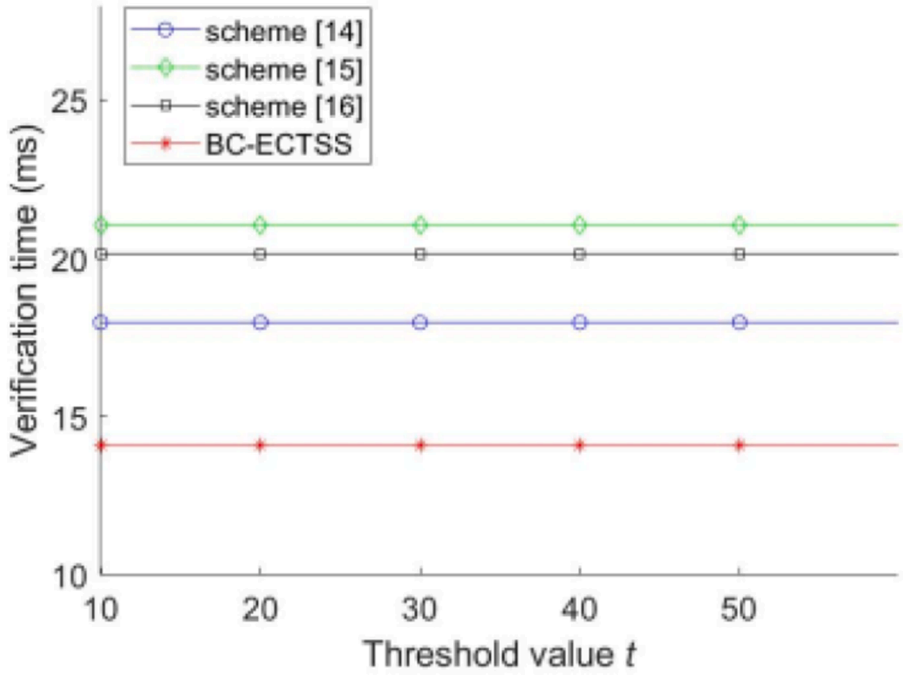

Fig. 8. Verification time comparison.

Declaration of Competing Interest

Authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this artical.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the Key Project of Natural Science Foundation of Shannxi Province under Grant 2020JZ-54.

References

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. Decentral Busi Rev 2008.
[2] Schinckus C. The good, the bad and the ugly: An overview of the sustainability of blockchain technology. Energ Res Soc Sci 2020;69:101614.

[3] Dai H, Zheng Z, Zhang Y. Blockchain for Internet of Things: A survey. IEEE Internet Things J 2019;6(5):8076–94.
[4] Lu J, Wu S, Cheng H, et al. Smart contract for electricity transactions and charge settlements using blockchain. Appl Stoch Models Bus Ind 2021;37(3):442–53.
[5] Jia X, Lu Z, Wei H. Study on game soft product copyright protection based on blockchain. Appl Res Comp 2020;37(12):3691–8.
[6] Shamir A. How to share a secret. Commun ACM 1979;22(11):612–3.
[7] Stadler M. Publicly verifiable secret sharing. International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer; 1996. p. 190–9.
[8] Schoenmakers B. A simple publicly verifiable secret sharing scheme and its application to electronic voting. Annual International Cryptology Conference. Berlin: Springer; 1999. p. 148–64.
[9] Goldfeder S, Bonneau J, Kroll J, et al. Securing bitcoin wallets via threshold signatures. 2014.
[10] Yan J, Lu Y, Chen L, et al. A SM2 elliptic curve threshold signature scheme without a trusted center. KSII Trans Intern Inform Syst (TIIS), 10(2): 897-913.
[11] Goldfeder S, Gennaro R, Kalodner H, et al. Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme. 2015.
[12] Cheng Y, Jia Z. Threshold Signature Scheme for Electronic Voting Scenarios. J Comp Appl 2019;39(09):2629–35.
[13] Yu H, Wang S. Certificateless threshold signcryption scheme with secret sharing mechanism. Knowl-Base Syst 2021;221:106981.
[14] Dalskov A, Orlandi C, Keller M, et al. Securing DNSSEC keys via threshold ECDSA from generic MPC. In: European Symposium on Research in Computer Security. Springer, Cham; 2020. p. 654–73.
[15] Gennaro R, Goldfeder S. Fast multiparty threshold ECDSA with fast trustless setup. In: 2018 ACM SIGSAC Conference on Computer and Communications Security; 2018. p. 1179–94.
[16] Aumasson J, Hamelink A, Shlomovits O. A survey of ECDSA threshold signing. Cryptology ePrint Archive 2020.
[17] Doerner J, Kondi Y, Lee E, et al. Secure two-party threshold ECDSA from ECDSA assumptions. IEEE Symposium on Security and Privacy. IEEE; 2018. p. 980–97.
[18] Yang B. Modern cryptography. BeiJing: Tsinghua University Press; 2015. p. 65–9.
[19] Zhou J, Qu R, Sun L. Efficient blockchain wallet protection scheme against single point failure. Appl Res Comp 2021;38(07):1947–51.
[20] Wang T, HOU S. Research on threshold signature scheme and its security analysis. Comp Eng Appl 2018;54(13):123–30.
[21] Zhang S. A threshold signature scheme based on self-bilinear pairing algorithm over elliptic curves. J Hainan Norm Univ 2019;32(02):193–8.
[22] Al-Riyami S, Paterson K. Certificateless public key cryptography. International conference on the theory and application of cryptology and information security 2003:452–73.