



Blockchain-based two-party fair contract signing scheme

Liang Zhang^{a,b,1}, Hanlin Zhang^{a,c,1}, Jia Yu^{a,d,e,*}, Hequn Xian^{a,b}

^a School of Computer Science and Technology, Qingdao University, Qingdao 266000, China

^b State Key Laboratory of Integrated Services Networks (Xidian University), Xian 710071, China

^c Business School, Qingdao University, Qingdao 266000, China

^d State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

^e State Key Laboratory of Cryptography and Science, Beijing 100878, China

ARTICLE INFO

Article history:

Received 3 September 2019

Received in revised form 14 January 2020

Accepted 13 May 2020

Available online 21 May 2020

Keywords:

Ethereum

Smart contract

Blockchain

Fair contract signing

ABSTRACT

A fair contract signing scheme ensures that contract signing participants can fairly exchange the digital signatures of the contract, which has a wide range of applications on the Internet. Verifiable Encrypted Signature (VES) can be used as a fair exchange mechanism for digital signatures, in which it requires a centralized Trusted Third Party (TTP) as an adjudicator. However, there are many security challenges with the centralized TTP. For example, it may disclose the contents of the contract, collude with others, or even occurs a service interruption. To address those problems, in this paper, we propose a two-party fair contract signing scheme based on Ethereum smart contract, which allows participants to fairly perform the contract signing procedures on the blockchain. Specifically, we propose a modified VES scheme in which no centralized adjudicator is needed. Then we design the fair contract signing scheme based on the modified VES scheme. We leverage the Ethereum smart contract technology to ensure fairness, which has the characteristics of decentralization, verifiability, autonomy, efficiency, and tampering resistant. The theoretical analysis and experimental results show that our scheme is secure and feasible.

© 2020 Elsevier Inc. All rights reserved.

1. Introduction

Nowadays, the Internet is developing rapidly, signing an electronic contract online has become a common commercial activity. One crucial security requirement of signing contract online is that the contract signing participants can exchange their digital signatures fairly [17]. The fairness means that both participants could get a valid signature of the other party or neither of them can get it. However, the process of contract signing is asynchronous, which inevitably brings a certain degree of unfairness [31]. That is, after one participant received the signature of the other participant, he/she may not submit their signatures to the other party intentionally or refuse to admit that he/she has received the signature. The unfairness will hurt the rights of participants and impede the healthy development of the online commercial activities, such as payment services for secure outsourcing mechanism [24,35,34]. Hence, a fair contract signing scheme is very essential to guarantee fairness in the electronic contract signing process.

The VES [4] is a cryptographic algorithm that can be used to achieve fair exchange of digital signatures. The VES enables the verifier to check the validity of an encrypted signature for a particular message. However, the VES often requires a

* Corresponding author at: School of Computer Science and Technology, Qingdao University, Qingdao 266000, China.

E-mail address: qduyujia@gmail.com (J. Yu).

¹ Liang Zhang and Hanlin Zhang contributed equally to this work.

centralized TTP as an adjudicator to ensure the fairness of signatures exchange, which causes some security problems [26]. First, most third parties are highly centralized. They can obtain some sensitive information on the contract, including the details of the contract, the digital signatures of the contract, etc. More worse, they might disclose these confidential data for financial incentives. Second, the third party might be dishonest and collude with one participant, which would cause financial losses for the other participant. Third, the third party service can be terminated due to software and hardware fails, which would cause serious losses for both contract signing participants. These problems will become bottlenecks of fair signatures exchange. How to fairly exchange signatures without a third party is a critical and unsolved problem.

In recent years, the development of blockchain technology provides the possibility to solve the above problem. Intuitively, blockchain can be seen as a decentralized TTP, which can eliminate the security problems of a centralized TTP. The smart contract technology based on Ethereum [8] has aroused widespread attention from researchers. Ethereum is a global decentralized TTP, the dynamic joining of nodes can ensure that single-node failure has little effect on the whole network. Meanwhile, Ethereum-based smart contract technology has the characteristics of public verifiability, autonomy, high efficiency, and tampering resistant, which can be used to achieve decentralized fairness. Therefore, designing a fair mechanism based on the Ethereum smart contract technology can effectively ensure the fairness of signatures exchange in the process of signing an electronic contract.

In this paper, we propose a modified VES scheme in which no centralized TTP is required. Based on this VES scheme, we propose a two-party fair contract signing scheme which satisfies the following properties:

- (1) Fairness: Our scheme can guarantee that either both contract signing participants will get the ordinary signature of the other party or both of them will not get the ordinary signature of the other party. The fairness is mainly based on the over-time penalty mechanism and the credit mechanism which are implemented based on the Ethereum smart contract.
- (2) Privacy: In our proposed scheme, sensitive information in the contract will be kept so secret that it does not expose to blockchain nodes. Thus important contract content is only known to both contract signing participants. Besides, our scheme will ensure that only contract signing participants can extract the ordinary digital signature of the other party.
- (3) Security: Our scheme is secure against existential forgery and extraction. Moreover, Ethereum blockchain can resist single-node failures. That will not terminate the signature service due to partial node failures. Therefore, our scheme has high security and reliability.

The remainder of this paper are organized as follows. Section 2 briefly introduces some preliminary knowledge; Section 3 describes the system model of our scheme and illustrates our proposed scheme in detail; We analyze the correctness and security of our scheme in Section 4; Then, we introduce the system implementation and performance evaluation in Section 5 and Section 6 respectively; We review related work in Section 7; Finally, Section 8 concludes this paper.

2. Preliminaries

In this section, we briefly introduce the background knowledge, including the blockchain technology, the Ethereum smart contract, and the related cryptography knowledge.

2.1. Blockchain technology and Ethereum smart contract

Blockchain technology is the core technology of Bitcoin and has recently become a disruptive innovation with a wide range of applications [27]. Bitcoin was proposed by a scholar named “Satoshi Nakamoto” [18] in 2008. In his white paper, Bitcoin is the first electronic payment system that does not require a TTP in a decentralized P2P transaction network. The underlying blockchain technology is defined as a decentralized distributed shared ledger. Fig. 1 shows the block structure of the Bitcoin blockchain. Each block consists of two parts: the block header and the block body. The block header contains six attributes. Each part plays an important role in the generation of the block. A complete list of transactions is saved in the block body. The special block structure of the blockchain guarantees that the transaction data is tamper-resistant, unforgeable and traceable.

The general concept of smart contract was first proposed by Nick Szabo [23], who defined smart contracts as: “Smart contracts contain some promises generated in the form of digitalization, including protocols on which contract participants can execute these commitments.” Specifically, blockchain-based smart contracts are a computer protocol. That are used to broadcast, verify, and execute contracts by an informational form. Smart contracts can execute and complete trusted transactions without a TTP, which are traceable and irreversible [9]. Ethereum [8] is the first open smart contract development platform that allows developers to design and run decentralized applications through smart contracts.

2.1.1. Solidity and Ethereum virtual machine (EVM)

Ethereum provides a Turing-complete programming language called Solidity [3]. The smart contract written in Solidity can be compiled into EVM bytecode, which will be executed on the EVM [12]. The EVM is a smart contract execution environment, which is running on all Ethereum nodes. The EVM is a completely independent sandbox, the contract code is executed automatically and correctly without interference from the external environment. To ensure that EVM resources are

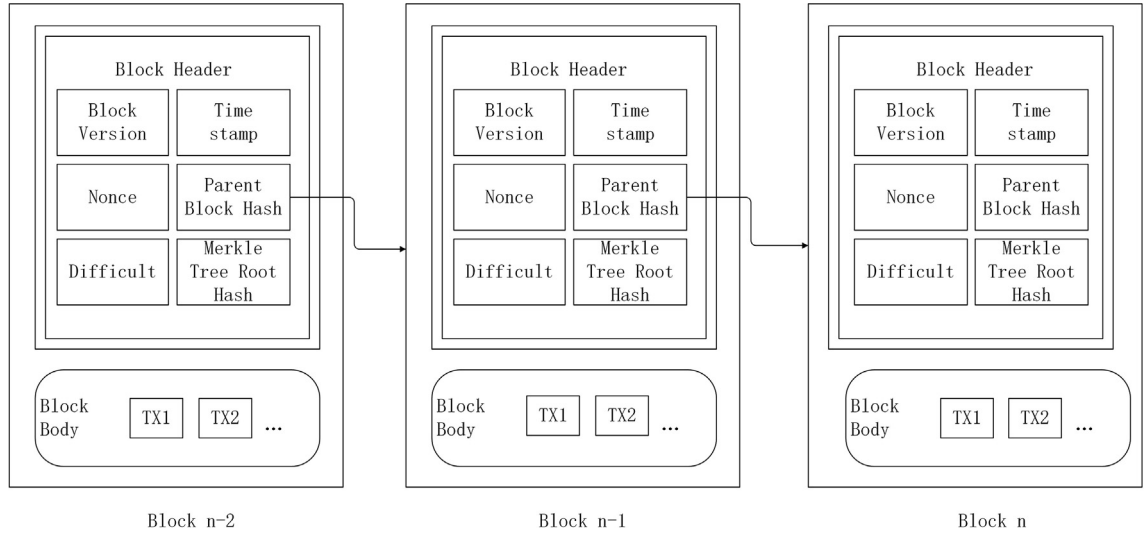


Fig. 1. Structure of Bitcoin block.

properly utilized, every instruction is at a certain cost. This encourages developers to write high-quality code to lower the transaction costs [30].

2.1.2. The full lifecycle of smart contract

The smart contract life cycle consists of three parts [11]: smart contract generation, smart contract release, and smart contract execution, as shown in Fig. 2.

- (1) Smart contract generation: Smart contract participants negotiate and program the smart contract code. Standardized smart contract code will be formed after formal verification.
- (2) Smart contract release: Similarly to the issuance of a transaction, the Ethereum nodes package the recently released smart contracts into a set of smart contracts, then store them in a block after multiple forwardings and verification.
- (3) Smart contract execution: When the smart contract receives the trigger condition, the smart contract code will be executed automatically. If no node is malicious, each node will get the same result. The entire network nodes verify the result and save it to the block.

2.2. Cryptography knowledge

2.2.1. Bilinear pairing

Let G_1 be an additive cyclic group. Let G_2 be a cyclic multiplication group, and the order of G_1 and G_2 is a prime p . P is a random generator of G_1 . Defining $e : G_1 \times G_1 \rightarrow G_2$ as a bilinear map. It satisfies the following conditions:

- (1) Bilinear: For $\forall Q, S \in G_1, a, b \in \mathbb{Z}_p^*$, $e(aQ, bS) = e(Q, S)^{ab}$.
- (2) Non-degenerate: $\exists Q, S \in G_1$ such that $e(Q, S) \neq 1$.
- (3) Computability: For $\forall Q, S \in G_1$, there is a valid polynomial time algorithm to calculate $e(Q, S)$.

The points on an elliptic curve form an additive group, and G_1 is a subgroup of this additive group. G_2 is a subgroup of the multiplicative group on a finite field. The groups G_1 and G_2 described below are based on this definition.

2.2.2. Strong Diffie-Hellman assumption

The definition of r -strong diffie-hellman (r -SDH) problem [6] in the groups (G_1, G_2) is as follows: Taking a $(r + 1)$ tuple $(P, aP, a^2P, \dots, a^rP)$ as input, compute $(c, \frac{1}{a+c}P)$, where $c \in \mathbb{Z}_p^*$. There is an advantage ϵ to deal with r -SDH in G_1 by using an algorithm A if.

$$\text{Proof}\{A(P, aP, a^2P, \dots, a^rP) = (c, \frac{1}{a+c}P)\} \geq \epsilon$$

r - SDH Assumption. The probability of dealing with the r -SDH problem is negligible for any polynomial time algorithms [7].

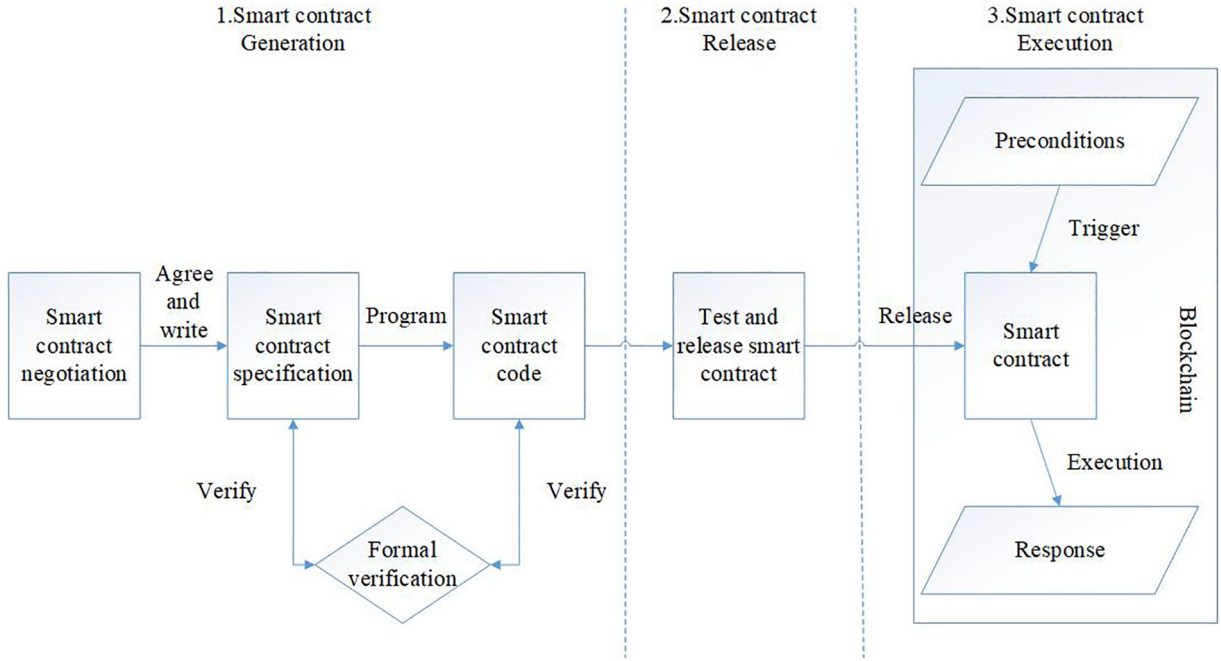


Fig. 2. The full lifecycle of smart contract.

2.2.3. Basic VES scheme

According to the Definition 3 in [7], a basic VES scheme contains three entities: the Signer, the Verifier, and the Adjudicator. The eight algorithms are as follows: Setup, KeyGen, Sign, Verify, AdjKeyGen, VESigCreate, VESigVerify, Adjudicate. The first four algorithms are not different from the ordinary digital signature scheme. The others provide the VES capability.

Setup: Generate the system parameters: $params$.

KeyGen: Input system parameters $params$, and output the signer's public-private key pair (PK, SK) .

Sign: Input the signer's private key SK and message M , and output the signer's ordinary signature σ .

Verify: Input the signer's ordinary signature σ , public key PK and message M . Verify if the signature σ is correct, then output "true"; otherwise output "false".

AdjKeyGen: Input system parameters $params$, and output the adjudicator's public-private key pair (APK, ASK) .

VESigCreate: Input the signer's private key SK , message M and the adjudicator's public key APK , and output the VES σ_{VES} on message M .

VESigVerify: Input the signer's public key PK , message M , the adjudicator's public key APK , and the VES σ_{VES} . Verify if σ_{VES} is a valid VES on message M under public key PK , then output "true"; otherwise output "false".

Adjudicate: Input the signer's public key PK , the VES σ_{VES} on message M and the adjudicator's public-private key pair (APK, ASK) . Extract and output the ordinary digital signature σ on M under public key PK .

3. Proposed scheme

In this section, we first give system model of our proposed scheme. Then we describe framework and introduce our design rationale about our scheme. Finally we describe our scheme in detail.

3.1. System model

The system model of our scheme is shown in Fig. 3. The two-party fair contract signing scheme based on the Ethereum smart contract consists of three entities: the contract signing parties Alice and Bob, and the Ethereum blockchain verification nodes. The procedures of Alice and Bob signing an electronic contract are described as follows.

First, Alice and Bob generate their own signature key pair (PK, SK) off-blockchain. The key pair needs to be authenticated by the authoritative authentication center and bound with the user's identity information. After both signing parties check the other identity information, the contract content and the related data will be negotiated. At the same time, to protect the sensitive information in the contract, both signing parties have to use a shared encryption key k to execute encryption operations for the sensitive information in the contract. Then the related data, such as the public key of both signing parties, the

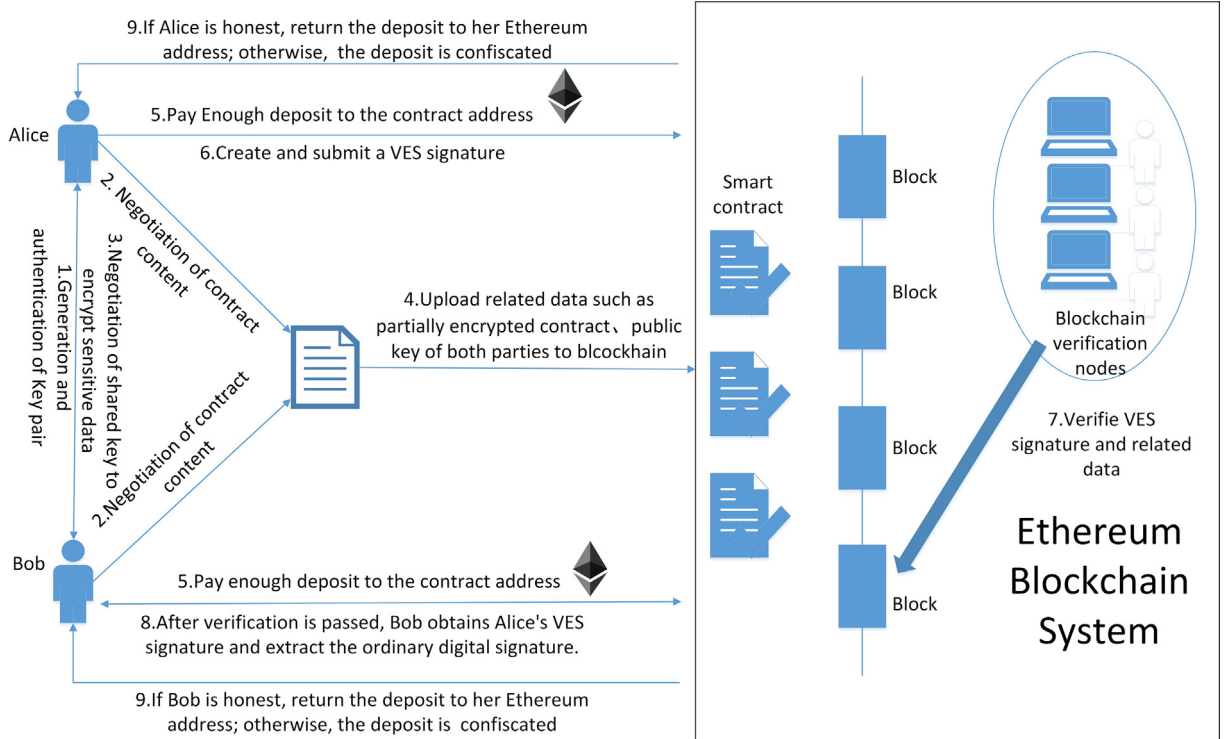


Fig. 3. System model.

encrypted contract, will be uploaded to the blockchain by the smart contract. After checking the data on the blockchain, Alice and Bob need to make sufficient deposits to the smart contract account before the specific time.

Next, after both signing parties make the deposits, Alice and Bob create their VES respectively off-blockchain. Then they submit their own VES and the related data to the blockchain by the smart contract. On receiving the signatures, a Verify smart contract, which is used to verify the validity of submitted VES, will be triggered. The Ethereum blockchain verification nodes run the smart contract code in local EVM to verify the validity of the submitted VES and the related data. Now suppose Alice's VES verification passes, Bob will be able to obtain Alice's VES and extract the ordinary signature from Alice's VES by using his private key.

Last, if both signing parties submit the valid VESes before a specified time, the contract is signed successfully. The deposits paid by both parties will be returned to their Ethereum addresses. Otherwise, an overtime penalty mechanism and a credit mechanism will be triggered to punish the dishonest party. The party that undermines the fairness of contract signing not only confiscates the deposit paid, but also reduces his credit value. After the above procedures, our scheme will be successfully completed.

3.2. Framework

Our scheme consists of eight algorithms: *Setup*, *KeyGen*, *ConNeg*, *Deposit*, *VES-Creation*, *VES-Verification*, *Sig-Extraction*, and *Judgment*.

Setup: Initialize system parameters.

KeyGen: Generate Alice's public-secret key pairs (PK_a, SK_a) and Bob's public-secret key pairs (PK_b, SK_b) .

ConNeg: Alice and Bob negotiate the contract m and the secret key k which will be applied to encryption operations off the blockchain, then submit the encrypted contract c and the related data to the blockchain.

Deposit: Alice and Bob make sufficient deposits to the smart contract. If both parties succeed in paying the deposits before the specific time, the contract signing will continue; otherwise, the contract signing will be terminated.

VES-Creation: For the contract c , Alice selects a random r and inputs $(r, SK_a, PK_b) \rightarrow \langle r, \sigma_{VES-Alice} \rangle$.

VES-Verification: Alice submits $(\langle r, \sigma_{VES-Alice} \rangle, PK_a, PK_b) \rightarrow (ture, false)$.

Sig-Extraction: Bob inputs $(\langle r, \sigma_{VES-Alice} \rangle, SK_b) \rightarrow \sigma$.

Judgment: Based on the result of signature verification, if both parties are honest, deposits paid by them will return their Ethereum addresses; otherwise, if dishonest behavior exists, dishonest party will be punished according to the designed penalty mechanism.

3.3. Design rationale

The goal of our scheme is to achieve fair execution of contract signing without a centralized TTP. We use Ethereum smart contracts to design an over-time penalty mechanism and a credit mechanism for ensuring fairness. Specifically, the contract signing participants are required to pay sufficient deposits before the contract is signed. Then the smart contract will automatically determine whether the participants are honest according to the result of the VES verification. Besides, since contract signing is executed on blockchain which is public, we should take the privacy into concern. First, sensitive information in the contract should be protected. Thus, we require the contract signing participants to encrypt the sensitive information in the contract with a shared key. Second, our scheme is based on the VES scheme proposed by Gorantla and Saxena [10]. There is no third party that can extract ordinary signatures. Thus, we designed the signing scheme which will ensure that only the recipient of the VES can extract the ordinary signature of the other party.

3.4. Detailed scheme

We now describe the eight procedures of our scheme in detail.

Step 1. Setup: Let (G_1, G_2) be bilinear groups. Their order is p . Define $e : G_1 \times G_1 \rightarrow G_2$ as a bilinear map and define an anti-collision hash function $H : \{0, 1\}^* \rightarrow Z_p^*$.

Step 2. KeyGen: Alice and Bob randomly choose $x, y \in Z_p^*$, then compute $u = xP, v = yP \in G_1$, and $z = e(P, P) \in G_2$. Alice's private key is (x_a, y_a) and the public key is (P, u_a, v_a, z) . Bob's private key is (x_b, y_b) and the public key is (P, u_b, v_b, z) .

Step 3. ConNeg: Alice and Bob negotiate the contract m to be signed and exchange their public key and Ethereum addresses for identity verification. Then the contract is set an index ID . Also, the sensitive content in the contract will be protected by generating a shared secret key k . Then, Alice and Bob use the secret key k to execute encryption operations for sensitive content. The encrypted contract is denoted by $c \in Z_p^*$. Next, the hash of the contract $H(m)$ and the hash of the partially encrypted contract $H(c)$ are calculated. $ID, H(m), c, H(c)$, the public key of both parties and the Ethereum addresses are uploaded to Ethereum blockchain by using the smart contract we designed. The blockchain nodes confirm this transaction by the consensus mechanism and then package them into the block which is linked to the blockchain.

Step 4. Deposit: After the contract negotiation and the related data upload are completed, the contract signing participants need to make sufficient deposits to the smart contract address. Deposits are mainly used to punish the party who acts dishonestly. If neither party make the deposits before the specified time, the contract signing will be terminated. If one party fails to pay the deposits before the specified time, the contract signing will also be terminated and the deposits paid will be refunded.

Step 5. VES-Creation: Alice and Bob sign the contract c for the VES respectively. Alice uses her private key (x_a, y_a) and Bob's public key (P, u_b, v_b, z) to generate a VES for the contract c , as shown below:

- Choose a random number $r \in Z_p^*$.
- Compute $\sigma_{VES-Alice} = \frac{1}{x_a + c + ry_a}(u_b + rv_b)$.

Alice's VES for contract c is $\langle r, \sigma_{VES-Alice} \rangle$, which is then submitted to the blockchain by smart contract along with Alice's public key and Bob's public key. Similarly, Bob generates a VES for the contract c by using his private key (x_b, y_b) and Alice's public key (P, u_a, v_a, z) , as shown below:

- Choose a random number $r' \in Z_p^*$.
- Compute $\sigma_{VES-Bob} = \frac{1}{x_b + c + r'y_b}(u_a + r'v_a)$.

Bob's VES for contract c is $\langle r', \sigma_{VES-Bob} \rangle$, which is then submitted to the blockchain by smart contract.

Step 6. VES-Verification: On receiving the VESes for the contract c and the related data from Alice and Bob, the smart contract running on the local EVM at the blockchain nodes verifies the validity of the their VESes. To prevent both sides from tampering with the sensitive information in encrypted contract, the smart contract first hashes the contract c to determine if it matches $H(c)$ which was stored previously. Then smart contract verifies Alice's VES $\langle r, \sigma_{VES-Alice} \rangle$ of the contract c with Alice's public key (P, u_a, v_a, z) and Bob's public key (P, u_b, v_b, z) by verifying the following equation. Alice's VES is valid if the equation holds, otherwise it is invalid:

$$e(\sigma_{VES-Alice}, u_a + cP + rv_a) = e(u_b + rv_b, P).$$

For Bob's VES $\langle r', \sigma_{VES-Bob} \rangle$ on the contract c , smart contract verifies its validity with Bob's public key (P, u_b, v_b, z) and Alice's public key (P, u_a, v_a, z) by verifying the following equation. Bob's VES is valid if the equation holds, otherwise it is invalid:

$$e(\sigma_{VES-Bob}, u_b + cP + r'v_b) = e(u_a + r'v_a, P).$$

If both of the VESes from Alice and Bob are valid, the scheme moves to step 7. Otherwise, the scheme moves to step 8 and the penalty mechanism will be triggered.

Step 7. Sig-Extraction: After the VES of Alice or Bob passes the verification, the other party can extract the ordinary signature. When Alice's VES passes the verification, Bob can use his own private key (x_b, y_b) to extract Alice's ordinary signature as follows:

$$\sigma_{Alice} = \frac{1}{x_b + ry_b} \sigma_{VES-Alice}.$$

When Bob's VES passes the verification, Alice can use her own private key (x_a, y_a) to extract Bob's ordinary signature as follows:

$$\sigma_{Bob} = \frac{1}{x_a + ry_a} \sigma_{VES-Bob}.$$

Step 8. Judgment: Based on the verification results, the corresponding actions will be taken. To ensure the fairness, we design an over-time penalty mechanism and a credit mechanism.

(a) Over-time penalty mechanism

We set a deadline for both parties to pass the verification on the VES. The behaviors of Alice and Bob can be categorized into the following three cases:

- Both of the VESes from Alice and Bob pass the verification before the deadline.
- Neither of them passes the verification when deadline is exceeded.
- Only one party passes the verification before deadline.

In the first case, the contract is successfully signed. The deposits will be returned to the Ethereum addresses of Alice and Bob. In the second case, the contract fails to be signed, the deposits will be returned to their Ethereum addresses, and the records will be left on the blockchain. Finally, the scheme is terminated. In the third case, the deposits paid by both parties will be transferred to the honest party's Ethereum address. Notice that although the deposit of the dishonest party is confiscated, the dishonest party still obtains the correct digital signature of the honest party. Thus, our scheme will take actions to make this contract invalid. First, after receiving the deposits of the dishonest party, the honest party publishes the secret key k and unencrypted contract m negotiated by both parties. The smart contract computes the hash value of m to obtain $H(m)$ and verifies if it equals $H(m)$. If the two hash values are equal, the smart contract marks the unencrypted contract m and the encrypted contract c signed by the honest party as invalid contracts.

(b) Credit mechanism

In addition to confiscating the deposits paid by the dishonest party, we also introduces a credit mechanism, which records the credit of each contract signing participant. When the party acts dishonestly, the smart contract reduces his credit. In the future, if a participant demands to sign a contract with another person, he can check the credit of this person beforehand. This mechanism can effectively reduce the possibility of participants making dishonest behavior.

4. Correctness and security analysis

In this section, we analyze the correctness and security of our scheme. Then, we also discuss the security of the proposed scheme from the perspective of blockchain technology security.

4.1. Correctness

First, we analyze the correctness of the VES verification process. Notice that the blockchain verifies Alice's VES by checking if $e(\sigma_{VES-Alice}, u_a + cP + rv_a) = e(u_b + rv_b, P)$, in which

$$\begin{aligned} & e(\sigma_{VES-Alice}, u_a + cP + rv_a) \\ &= e\left(\frac{1}{x_a + c + ry_a} (u_b + rv_b), x_aP + cP + ry_aP\right) \\ &= e\left(\frac{1}{x_a + c + ry_a} (u_b + rv_b), (x_a + c + ry_a)P\right) \\ &= e(u_b + rv_b, P)^{\frac{1}{x_a + c + ry_a} \times (x_a + c + ry_a)} \\ &= e(u_b + rv_b, P) \end{aligned}$$

Thus, the VES verification is correct. Then, we analyze the correctness of the signature extraction. Bob uses his own private key to extract Alice's ordinary signature from Alice's VES as $\sigma_{Alice} = \frac{1}{x_b + ry_b} \sigma_{VES-Alice}$, in which

$$\begin{aligned} \sigma_{Alice} &= \frac{1}{x_b + ry_b} \sigma_{VES-Alice} \\ &= \frac{1}{x_b + ry_b} \times \frac{1}{x_a + c + ry_a} (u_b + rv_b) \\ &= \frac{1}{x_b + ry_b} \times \frac{1}{x_a + c + ry_a} (x_b + ry_b P) \\ &= \frac{1}{x_a + c + ry_a} P \end{aligned}$$

The above equation implies $\sigma_{Alice} = \frac{1}{x_a + c + ry_a} P$. Next, we will prove that the extracted Alice's ordinary signature is correct, as follows

$$\begin{aligned}
& e(\sigma_{\text{Alice}}, u_a + cP + r v_a) \\
&= e\left(\frac{1}{x_a + c + ry_a} P, u_a + cP + r v_a\right) \\
&= e\left(\frac{1}{x_a + c + ry_a} P, (x_a + c + r x_a)P\right) \\
&= e(P, P)
\end{aligned}$$

Thus, the signature extraction is correct. Stated thus, our VES scheme is correct.

4.2. Security

We now analyze the security of the proposed scheme. The following basic signature scheme refers to the short signature scheme in [6].

Assertion 1. *If the basic signature scheme is secure against existential forgery, our VES scheme is also secure against existential forgery.*

Proof. To prove the above assertion, we use the reduction to absurdity. That is, our VES scheme is secure against existential forgery, then the basic signature scheme is also secure against existential forgery. Suppose there is an adversary A , he can forge our VES scheme in probabilistic polynomial time with a non-negligible probability. Next, we can construct another adversary B by using A , such that B will successfully forge the basic signature in probabilistic polynomial time with non-negligible probability. \square

First, we adopt the security model in [6]. The adversary B generates a public-private key pair (SK, PK) for the VES scheme using the basic signature scheme, where $SK = (x', y')$, $PK = (P', u', v', z')$. The adversary B disguises as a contract signing participant. Next, B will execute the attack process based on the assumption that A can forge the VES. If A successfully forges the VES $(r', \sigma'_{\text{VES}})$ on a message m' , then B can generate a forged signature $\sigma' = \frac{1}{x' + r'y'} \sigma'_{\text{VES}}$ on a message m' from the forged $(r', \sigma'_{\text{VES}})$.

However, according to Theorem 3.1 in [6], under a chosen message attack, assuming r-SDH problem is hard, the base signature scheme is secure against existential forgery, so the VES in our scheme is unforgeable.

Assertion 2. *If the basic signature scheme is secure against existential forgery, no one but the receiver can extract the signature in our scheme.*

Proof. Only the signature receiver can extract the ordinary signature (r, σ) of message m from the VES (r, σ_{VES}) , and no adversary can extract the ordinary signature from it. On the basis of this, we can conclude that a VES on a message m is secure against extraction. Now suppose that there is an adversary A who wants to get the ordinary signature (r, σ) on the message m , so he either needs to forge it directly or extract it from the VES (r, σ_{VES}) . And the forged signature can pass the verification successfully. \square

According to the basic signature scheme in [6], directly forge an ordinary signature is almost impossible for adversary A . Hence, we now prove that A cannot extract the ordinary signature (r, σ) that satisfies the verification equation from the VES (r, σ_{VES}) . Alice's VES can be verified by following equation:

$$\begin{aligned}
& e(\sigma_{\text{VES-Alice}}, u_a + cP + r v_a) \\
&= e\left(\frac{1}{x_a + c + ry_a} (u_b + r v_b), (x_a + c + r y_a)P\right) \\
&= e(u_b + r v_b, P)^{\frac{1}{x_a + c + ry_a} \times (x_a + c + ry_a)} \\
&= e(u_b + r v_b, P) \\
&= e((x_b + r y_b)P, P) \\
&= e(P, P)^{x_b + ry_b}
\end{aligned}$$

Since the bilinear pair satisfies bilinearity, we next have.

$$e\left(\frac{1}{x_b + ry_b} \sigma_{\text{VES-Alice}}, u_a + cP + r v_a\right) = e(P, P)$$

Furthermore, from the above equation and the verification equation of the ordinary signature, we can infer that the ordinary signature is $\sigma = \frac{1}{x_b + ry_b} \sigma_{\text{VES-Alice}}$. If the adversary A wants to extract the ordinary signature (r, σ) , he must obtain the value of $x_b + r y_b$. This is equivalent to solving the discrete logarithm problem(DLP) with the base P in G_1 , which is considered to be computationally difficult. So adversary A cannot get the value of $x_b + r y_b$. Therefore, the VES in our scheme is secure against extraction.

Remark: We now analyze the security of our scheme from the perspective of blockchain technology. The security of blockchain technology is based on the cryptography algorithms and consensus mechanism. As the cryptographic algorithms applied in the blockchain have been proofed to be secure, the only considerable security concern is the so-called 51% attack. Theoretically, as long as the attacker controls more than 50% of the computation power, he can modify the data on the block-

chain. And there is no effective approach to deal with such attacks. However, in reality, the cost of the 51% attacks is too expensive. The attacks cannot make any profit with such expensive costs to launch the attack. So far, no such attacks have occurred. Hence, our scheme is still secure from the perspective of blockchain technology security.

5. System implementation

In this section, we first describe our simulated development environment of smart contracts. We then introduce some key functions in the smart contracts.

We develop the smart contracts with the Remix IDE running in Google Chrome. Remix [2] is an integrated development environment for solidity that integrates a debugger and test environment. It can deploy and test smart contracts in a web browser and no server component is required. Remix provides five test accounts with 100 Ether, we use account 1 as the initiating account for the smart contracts, account 2 and account 3 refer to Alice and Bob respectively.

Our scheme consists of ConNeg smart contract, Deposit smart contract, Verify smart contract and Judgment smart contract, which are described in detail in Tables 1–4.

Table 1 describes functions of ConNeg smart contract about data storage and query on the blockchain. The significance of storing data in blockchain is that the relevant data in the contract signing process will not be tampered and can be used in subsequent verification process. After contract signing participants successfully submit data to the blockchain, the hash value of this transaction will be returned. Both contract signing participants can query the data on the blockchain by the returned transaction hash value. We provide a data query interface, they can query the related data with the negotiated contract ID. Table 2 shows function of the Deposit smart contract. It handles the deposits from the contract signing participants. Table 3 describes the Verify smart contract, which handles the submission and verification of Alice and Bob's VESes. Notice that the current Ethereum smart contract lacks of a library of verification calculation for bilinear pairings, we simulate this process here with off-blockchain calculation instead of submitting calculation result to smart contract. Then, the Verify smart contract returns the signature verification result to the Judgment smart contract for corresponding actions. Table 4 provides some of the key functions description of Judgment smart contract. When unfair behavior occurs, the penalty mechanism designed by Judgment phase of our scheme will be triggered.

We now take the Deposit smart contract as an example to show the life cycle of our smart contracts. First, we program the smart contract in Solidity. Listing 1 shows the code of Deposit smart contract, which ensures that both contract signing participants will continue to perform the contract signing only when they have made the deposits before the deadline.

We deploy the smart contract using Remix. Fig. 4 shows the testing interface of smart contract. When the console outputs a transaction, it means our smart contract deployment is completed. Fig. 5 shows the initialization conditions required before the Deposit smart contract is deployed, including the agreed deadline for deposits. Fig. 6 is a screenshot of the query balance transaction, which shows specific transaction information. We can see that deposits are paid successfully by both parties.

Table 1
Functions of ConNeg smart contract.

Function	Description
setStorage	Enable data storage on the blockchain
getStorage	Enable data query on the blockchain

Table 2
Functions of deposit smart contract.

hline Function	Description
beforeDeadline (Modifier)	Determine whether the participant made deposits before deadline
afterDeadline (Modifier)	Determine whether the deposits deadline is exceeded
Deposit	Enable parties to make deposits
Refund	Enable the deposits refund
getBalance	Interface for querying contract address balance

Table 3
Functions of verify smart contract.

Function	Description
hashVerify	Verify the hash value of encrypted contract c
sigVerify	Verify the correctness of the VES

Table 4
Functions of judgment smart contract.

Function	Description
beforeEndtime (Modifier)	Determine whether the participant submitted the VES before endtime
afterEndtime (Modifier)	Determine whether endtime for VES submission is exceeded
depositRefund	Enable the deposits refund before the endtime
penalty	Punish the dishonest participant after the endtime
setCredit	Set credit value for contract signing participants based on their behavior
getCredit	Query credit value by identity
queryContract	Query contract signing result and the related information

Listing 1: Solidity code of Deposit smart contract

```

contract Deposit{
    uint depositGoal;//total deposit
    uint deadline;//end time
    uint nowDeposit;//deposit paid
    //record the address and amount of the payer
    mapping (address => uint) public balanceOf;
    //constructor
    function Deposit(uint _depositGoal, uint _runTime) public {
        depositGoal = _depositGoal *10**18;
        deadline = now + _runTime *1 min;
    }
    //Determine if the deadline has passed
    modifier afterDeadline() {
        if (now > deadline) _; } //Determine if the deadline has not passed
    modifier beforeDeadline() {
        if (now <= deadline) _; }
    //Deposit to smart contract
    function deposit() beforeDeadline payable public {
        require(nowDeposit < depositGoal);
        require(msg.value > 0);
        balanceOf[msg.sender] += msg.value;
        nowDeposit += msg.value;
    }
    //Query smart contract deposit balance
    function getBalance() constant returns(uint){
        return this.balance;
    }
    //Refund in case of non-payment of deposit
    function refund () afterDeadline public returns(bool){
        require(nowDeposit < depositGoal && nowDeposit > 0);
        uint amount = balanceOf[msg.sender];
        nowDeposit -= amount; balanceOf[msg.sender] = 0;
        return address(msg.sender).send(amount);
    }
}

```

6. Performance evaluation

In this section, we first show off-blockchain experiments of our VES scheme and subsequently evaluate the performance of our scheme by analysing the cost of gas when our scheme is executed by smart contract on the Ethereum blockchain.

We first present off-blockchain experiments as performance evaluation of the VES algorithm in our scheme. We built the testbed for the VES scheme on Ubuntu 16.04 system; The machine is with an *Inter^R CoreTMi5 – 3230M* CPU at 2.60 GHz, 4 GB in memory. The programming language we use is C++, and PBC (Pairing-Based Cryptography) library also is used to implement our VES scheme. The PBC [1] library is based on GMP library that executes the mathematical operations underlying pairing-based cryptosystems. Table 5 lists the function of the VES algorithm and their functional descriptions.

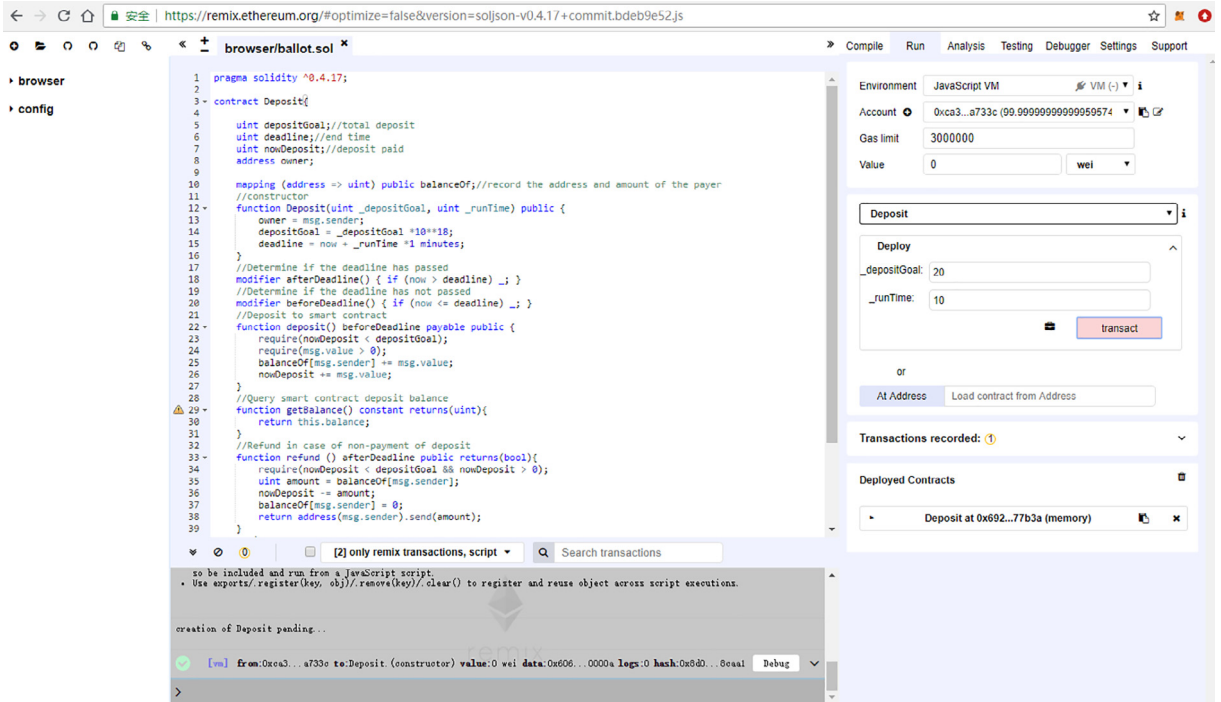


Fig. 4. Interface of testing deposit smart contract.

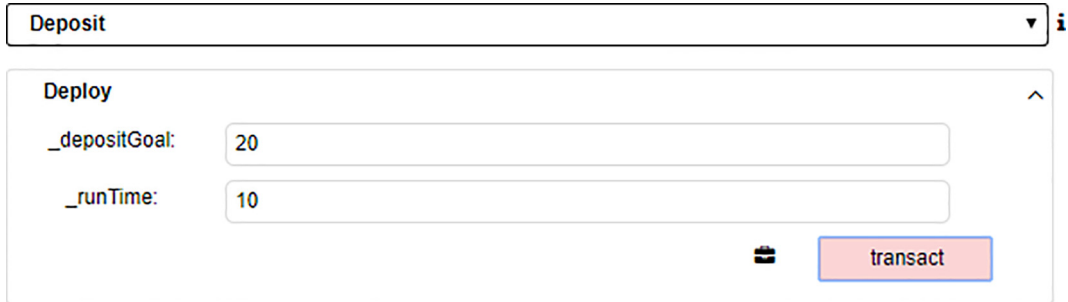


Fig. 5. Initialization conditions required before smart contract deployment.

In our experiment, we use a 256 KB contract document for testing. The size of prime p ranges from 160 bits to 1120 bits. To avoid the occasional data points, we execute the VES algorithm 100 times and record the average time cost. Fig. 7 shows the time costs of key generation, signature creation, signature verification and signature extraction for VES scheme, which gives us the following two observations. First, we can observe that the time cost in the key generation phase increases with the size of p . Notice that in our scheme, the calculations are in Z_p^* . Thus, the size of p decides the calculation costs. Second, the figure indicates that the average time cost in signature verification phase is higher than that in other phases. The signature verification phase requires three multiplication and two bilinear pairing on elliptic curves, which are time-consuming. Key generation phase costs longer time than the VES creation phase, as it involves four multiplication on elliptic curves. Signature creation phase only involves one modular inversion on Z_p^* and one multiplication on elliptic curves. Signature extraction phase costs the least time as it only involves one modular inversion on Z_p^* and one multiplication on elliptic curves.

To evaluate the performance of current smart contracts we designed on the Ethereum blockchain, we take the gas cost of executing smart contracts into consideration. Because the gas cost given in Remix is consistent with that in Ethereum blockchain, we adopt the gas cost estimated in Remix. The gas cost of partial function calls with relatively high cost in smart contracts is shown in Table 6. Transaction cost refers to the cost of sending a transaction to Ethereum blockchain. It is based on the size of data. Execution cost refers to the cost required by the EVM to execute computational operations. The total cost is the sum of the above mentioned two parts. We convert it to dollars by using an exchange rate of 1 Gas $\approx 4 \times 10^{-9}$ Ether and 1


 [call] from:0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db to:Deposit.getBalance() data:0x120...65fe0	
transaction hash	0xe9069ee73aa6cf7d4b59511490bf5cdf605fb08e2aa1edc0791571ac49e8642b
from	0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db
to	Deposit.getBalance() 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
transaction cost	21850 gas (Cost only applies when called by a contract)
execution cost	578 gas (Cost only applies when called by a contract)
hash	0xe9069ee73aa6cf7d4b59511490bf5cdf605fb08e2aa1edc0791571ac49e8642b
input	0x120...65fe0
decoded input	{}
decoded output	{ "0": "uint256: 20000000000000000000" }
logs	

Fig. 6. Transaction details about deposits paid successfully.

Table 5

Function of VES algorithm off the blockchain.

Function	Description
keyGen	Generate key pairs for the contract signing participants
vesCreation	Generate a VES for the encrypted contract off the blockchain
vesVerify	Verify submitted VES off the blockchain
vesExtract	Extract the ordinary signature from the VES after verification is passed

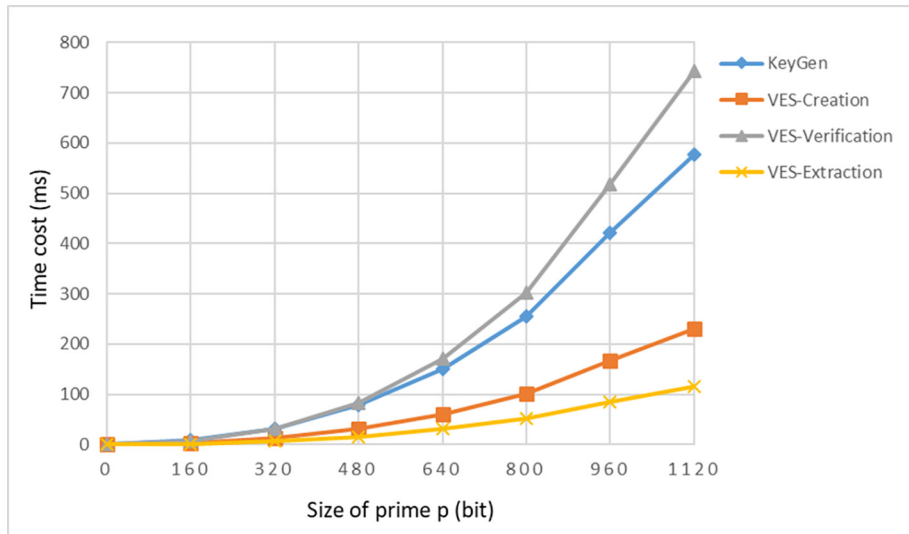


Fig. 7. Time spent on each phase of the VES scheme.

Ether \approx 130 dollars in January 2020. Based on Table 6, we can estimate the cost of our scheme roughly. When both contract signing parties sign a contract, they need to pay about \$0.45 for the whole procedure. After several rounds of testing and verification, we conclude that the cost of our scheme is low and acceptable.

Table 6

The gas cost of partial function calls in our scheme.

Function calls	Transaction cost (Gas)	Execution cost (Gas)	Total cost (\$)
setStorage	324020	284636	0.3164
deposit	62664	41392	0.0541
sigVerify	32884	1692	0.0179
depositRefund	22176	524	0.0118
penalty	33867	1715	0.0184
setCredit	41961	20305	0.0325

7. Related work

In this section, we introduce some related work about this paper from the two aspects: the VES schemes and fair blockchain-based protocols.

7.1. Verifiable encrypted signature schemes

Digital signatures are widely deployed in online contract signing. The VES signature is designed to serve the field of contract signing so that the fairness can be guaranteed. The concept of VES was first proposed in [4]. A few years later, Boneh et al. [7] described an aggregate VES scheme based on bilinear pairings. They showed that aggregate signatures give rise to the VES. And they explained how the VES can be used in contract signing protocols. Based on properties of bilinear pairings, Zhang et al. [33] proposed a new VES scheme. They analyzed the security and the efficiency about their scheme. Then, they also showed that their scheme is more efficient than those of the past. Similarly, using bilinear pairings, Gorantla et al. [10] proposed a VES scheme, which is secure against existential forgery under chosen message attack and extraction, without random oracles. Besides, Shao et al. [22] presented the first VES scheme by using discrete logarithm (DL). The security of this scheme in the random oracle model is based on the DL problem and the computational Diffie-Hellman (DH) problem. The VES scheme of Shao et al. is suitable for the Internet environment we are using at present. Other different VES schemes have also been proposed in [15,16,19,21]. All the above schemes need a TTP to ensure fairness. This will lead to security problems. How to achieve fair contract signing without a TTP is an unsolved critical problem before this work.

7.2. Fair blockchain-based protocols

There are some researches on fair protocols based on blockchain. In the field of secure computation, Bentov and Kumaresan [5] proposed a model of fairness, and showed that the fairness will can be achieve by using the Bitcoin network in secure computation. Kumaresan [14] studied a model for incentivizing correct computations. And he introduced a variety of cryptographic tasks where the model can be used. To address the fairness of the payment process, Wang et al. [29] proposed a new fair payment protocol in deduplication cloud storage based on the decentralization of blockchain. Security analysis and experimental result show that their protocol is feasible. Reddy et al. [20] designed two new ideal functionalities in Bitcoin and present a fair protocol for verifiable computation in the field of outsourcing computing. In addition, the fair contract signing scheme based on blockchain technology has become a research hotspot [25,28]. Based on the traditional idea of VES and blind signature, a blind VES scheme is constructed in [26]. However, the scheme uses the bitcoin transaction script to elaborate, which has certain limitations, and the verification process is easy to forge. On the basis of [26], Wu et al. [32] designed a new protocol by using the coin mixing technology to complete fair contract signing, which introduces a semi-honest third party for the mixing operations. Based on the primitive of blockchain, Hui et al. [13] proposed a fair three-party contract signing protocol, which gives a solution different from previous schemes. It allows for designing a fair protocol without a TTP. All the above protocols about contract signing are based on the Bitcoin network. The Bitcoin network does not provide a Turing-complete scripting language to implement complex logic. To fill this gap, we study to ensure the fairness of the contract signing process by Ethereum smart contract.

8. Conclusion

This paper proposes a two-party fair contract signing scheme based on Ethereum smart contract technology. The scheme uses automated smart contracts instead of the original TTP in the process of contract signing. In our scheme, the contract signing parties can exchange the signature fairly. Also, sensitive information in the contract will be kept so secret that it would not be exposed over the blockchain. Theoretical analysis and performance evaluation also show that our scheme is valid, secure and efficient. We now discuss some future work and related research directions. The shortcoming of our VES scheme in this paper is that the signature verification process cannot be directly completed by smart contract. When Ethereum smart contract is further improved and our VES scheme will can be fully deployed on the blockchain, a decentralized application of fair contract signing should be implemented and applied. Besides, there are many other blockchains, such as

the Hyperledger in the alliance blockchain, which has rich computing resources. In future research, we will consider improving our scheme to apply to other blockchains.

Declaration of Competing Interest

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

Acknowledgments

This research is supported by National Natural Science Foundation of China (61572267), National Development Foundation of Cryptography (MMJJ20170118), Key Research and Development Project of Shandong Province (2019GGX101051), the Open Project of the State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences (2019-MS-03).

References

- [1] Pbc library. <https://crypto.stanford.edu/pbc/>, 2019..
- [2] Remix, ethereum-ide. <https://remix.readthedocs.io/en/latest/>, 2019..
- [3] Solidity-solidity 0.5.7 documentation. <https://solidity.readthedocs.io/en/v0.5.7/>, 2019..
- [4] Nadarajah Asokan, Matthias Schunter, Michael Waidner, *Optimistic protocols for fair exchange*, Citeseer (1996).
- [5] Iddo Bentov, Ranjit Kumaresan, How to use bitcoin to design fair protocols, in: Proceedings of Annual Cryptology Conference, Springer, 2014, pp. 421–439..
- [6] Dan Boneh, Xavier Boyen, Short signatures without random oracles, in: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2004, pp. 56–73..
- [7] Dan Boneh, Craig Gentry, Ben Lynn, Hovav Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2003, pp. 416–432..
- [8] Vitalik Buterin, *A next-generation smart contract and decentralized application platform*, White Paper 3 (2014) 37.
- [9] Christopher D. Clack, Vikram A. Bakshi, Lee Braine, Smart contract templates: foundations, design landscape and research directions. arXiv preprint arXiv:1608.00771, 2016..
- [10] M. Choudary Gorantla, Ashutosh Saxena, Verifiably encrypted signature scheme without random oracles, in: Proceedings of International Conference on Distributed Computing and Internet Technology, Springer, 2005, pp. 357–363..
- [11] Haiwu He, An Yan, Zehua Chen, *Survey of smart contract technology and application based on blockchain*, J. Comput. Res. Dev. 55 (11) (2018) 112–126.
- [12] Yoichi Hirai, Defining the ethereum virtual machine for interactive theorem provers, in: Proceedings of International Conference on Financial Cryptography and Data Security, Springer, 2017, pp. 520–535..
- [13] Hui Huang, Kuan-Ching Li, Xiaofeng Chen, A fair three-party contract signing protocol based on blockchain, in: Proceedings of International Symposium on Cyberspace Safety and Security, Springer, 2017, pp. 72–85..
- [14] Ranjit Kumaresan, Iddo Bentov, How to use bitcoin to incentivize correct computations, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 30–41..
- [15] Lu. Steve, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, Brent Waters, *Sequential aggregate signatures, multisignatures, and verifiably encrypted signatures without random oracles*, J. Cryptol. 26 (2) (2013) 340–373.
- [16] Vadim Lyubashevsky, Gregory Neven, One-shot verifiable encryption from lattices, in: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2017, pp. 293–323..
- [17] Xiaoshu Ma, Yun Shang, *New scheme of fair exchange of digital signatures*, J. Comput. Eng. Appl. 44 (16) (2008) 98–99.
- [18] Satoshi Nakamoto, *Bitcoin: a peer-to-peer electronic cash system* (2008).
- [19] Ryo Nishimaki, Keita Xagawa, Verifiably encrypted signatures with short keys based on the decisional linear problem and obfuscation for encrypted vcs, in: Proceedings of International Workshop on Public Key Cryptography, Springer, 2013, pp. 405–422..
- [20] Dorsala Mallikarjun Reddy, V.N. Sastry, Chapram Sudhakar, Fair protocols for verifiable computations using bitcoin and ethereum, in: Proceedings of 2018 IEEE 11th International Conference on Cloud Computing, 2018, pp. 786–793..
- [21] Jae Hong Seo, Keita Emura, Keita Xagawa, Kazuki Yoneyama, Accumulable optimistic fair exchange from verifiably encrypted homomorphic signatures, Int. J. Inf. Security 17 (2) (2018) 193–220..
- [22] Zuhua Shao, Yipeng Gao, Practical verifiably encrypted signatures based on discrete logarithms, Security Commun. Networks 9 (18) (2016) 5996–6003.
- [23] Nick Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9) (1997).
- [24] Chengliang Tian, Yu. Jia, Hanlin Zhang, Haiyang Xue, Cong Wang, Kui Ren, *Novel secure outsourcing of modular inversion for arbitrary and variable modulus*, IEEE Trans. Services Comput. (2019).
- [25] Haibo Tian, Jiejie He, Liqing Fu, Contract coin: toward practical contract signing on blockchain, in: Proceedings of International Conference on Information Security Practice and Experience, Springer, 2017, pp. 43–61..
- [26] Haibo Tian, Jiejie He, Liqing Fu, A privacy preserving fair contract signing protocol based on block chains, J. Cryptol. Res. 4 (2) (2017) 187–198.
- [27] Sarah Underwood, *Blockchain beyond bitcoin* (2016).
- [28] Zhiguo Wan, Robert H. Deng, David Lee, Electronic contract signing without using trusted third party, in: Proceedings of International Conference on Network and System Security, Springer, 2015, pp. 386–394..
- [29] Shangping Wang, Yuying Wang, Yaling Zhang, *Blockchain-based fair payment protocol for deduplication cloud storage system*, IEEE Access 7 (2019) 127652–127668.
- [30] Maximilian Wöhler, Uwe Zdun, Smart contracts: security patterns in the ethereum ecosystem and solidity, in: Proceedings of 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), IEEE, 2018, pp. 2–8..
- [31] Jinxi Wu, Ying Gao, *Efficient multi party fair contract signing protocol based on blockchains*, J. Cryptol. Res. 5 (5) (2018) 556–567.
- [32] Wu. Jinxi, Ying Gao, Zongyang Zhang, Dapeng Yan, A multi-party privacy preserving fair contract signing protocol based on blockchains, J. Cyber Security 3 (3) (2018) 8–16.
- [33] Fangguo Zhang, Reihaneh Safavi-Naini, Willy Susilo, Efficient verifiably encrypted signature and partially blind signature from bilinear pairings, in: Proceedings of International Conference on Cryptology in India, Springer, 2003, pp. 191–204..
- [34] Hanlin Zhang, Jia Yu, Chengliang Tian, Guobin Xu, Peng Gao, Jie Lin, *Practical and secure outsourcing algorithms for solving quadratic congruences in internet of things*, IEEE Internet Things J. (2020).
- [35] Pu Zhao, Jia Yu, Hanlin Zhang, Zhan Qin, Cong Wang, How to securely outsource finding the min-cut of undirected edge-weighted graphs, IEEE Trans. Inf. Forensics Security 15 (2020) 315–328.